Reproducible projects in R: practical workshop

Marina Vabistsevits

What this workshop is about

1. **Project-oriented workflow** (organising your projects with .Rproj and renv)
   1.1. Rstudio efficiency tips
2. Using **Git** to track changes in your R projects
3. **R projects on UKB-RAP/All of Us** platforms with Git

Icon for R

Icon for RStudio

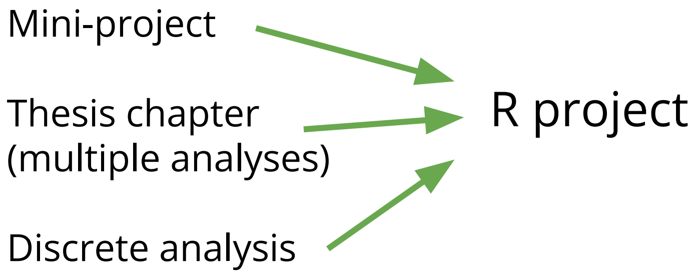
software company who develop Rstudio

Positron - new IDE from Posit for R/Python etc

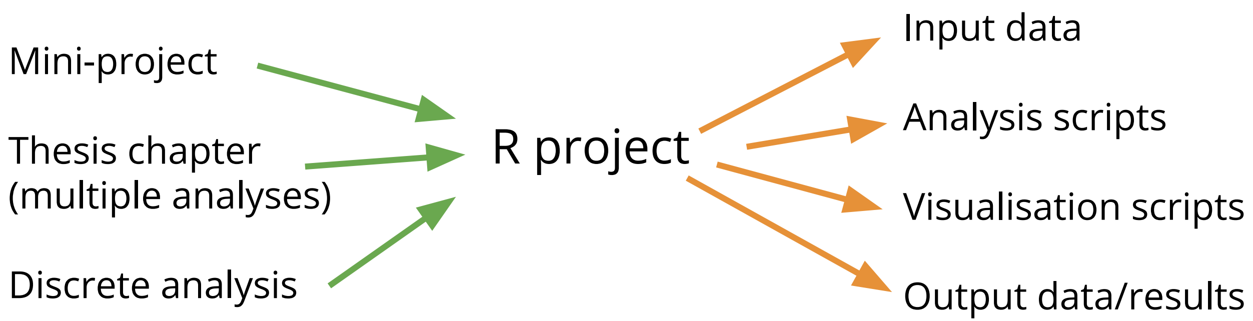
The slides will be shared after!

1. Project-oriented workflow

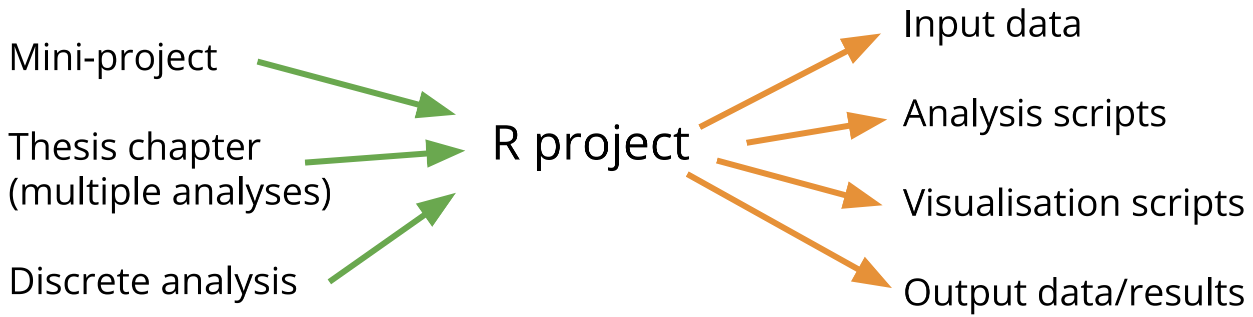
R projects


Mini-project
Thesis chapter
(multiple analyses)
Discrete analysis
R project

R projects


Mini-project
Thesis chapter
(multiple analyses)
Discrete analysis
R project
Input data
Analysis scripts
Visualisation scripts
Output data/results

R projects



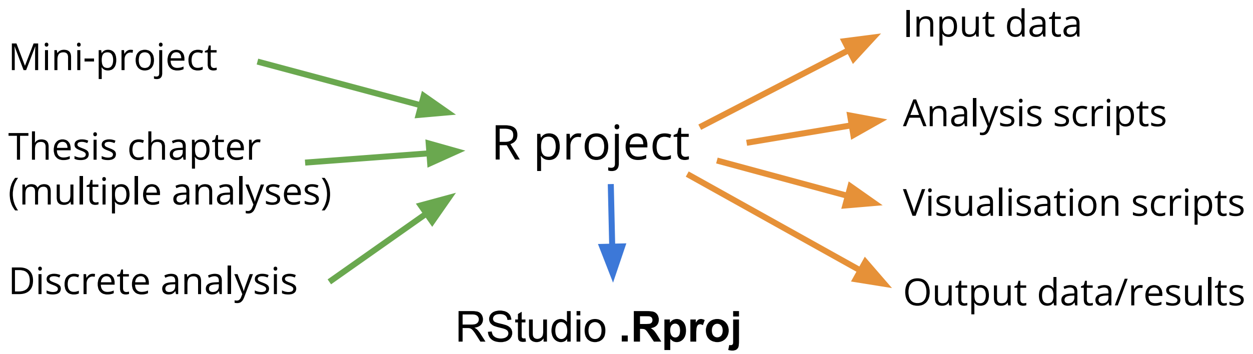
How you do organise your projects? scripts? folders?

Where do you store and access your data? results? figures?

Does it matter? Are your analysis/projects reproducible?

Can you organise your projects better and make life easier for future self/colleagues?

R projects



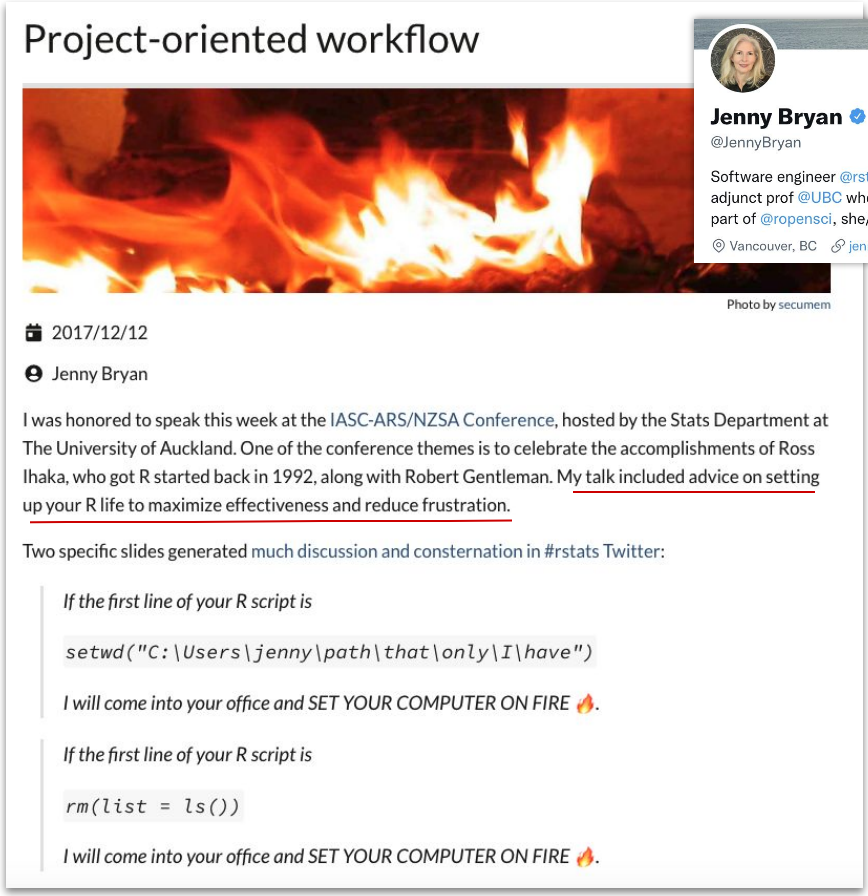
How you do organise your projects? scripts? folders?

Where do you store and access your data? results? figures?

Does it matter? Are your analysis/projects reproducible?

Can you organise your projects better and make life easier for future self/colleagues?

Reproducibility can be enhanced through intentionally organising projects with .Rproj, i.e. working in **project-oriented workflow**



Project-oriented workflow

Photo by secumem

2017/12/12

Jenny Bryan

I was honored to speak this week at the IASC-ARS/NZSA Conference, hosted by the Stats Department at The University of Auckland. One of the conference themes is to celebrate the accomplishments of Ross Ihaka, who got R started back in 1992, along with Robert Gentleman. My talk included advice on setting up your R life to maximize effectiveness and reduce frustration.

Two specific slides generated much discussion and consternation in #rstats Twitter:

> *If the first line of your R script is*
>
> `setwd("C:\Users\jenny\path\that\only\I\have")`
>
> *I will come into your office and SET YOUR COMPUTER ON FIRE* 🔥.

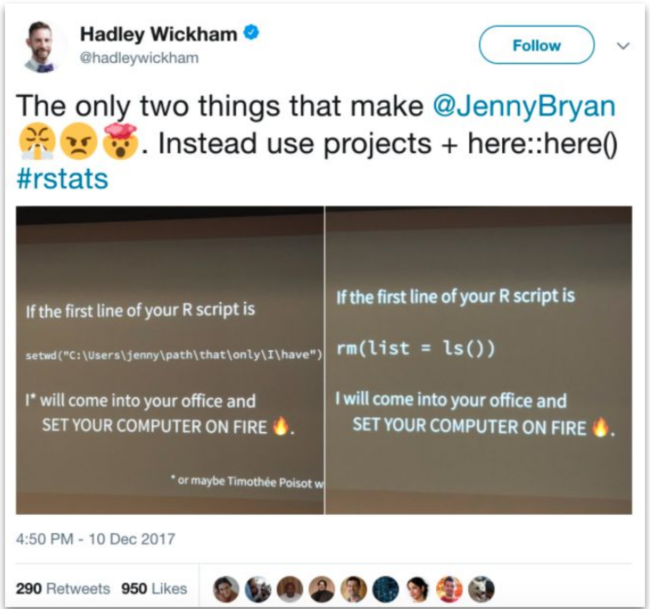
> *If the first line of your R script is*
>
> `rm(list = ls())`
>
> *I will come into your office and SET YOUR COMPUTER ON FIRE* 🔥.

Following

Jenny Bryan
@JennyBryan

Software engineer @rstudio, humane #rstats, adjunct prof @UBC where I created @STAT545, part of @ropensci, she/her

Vancouver, BC jennybryan.org



Hadley Wickham
@hadleywickham

Follow

The only two things that make @JennyBryan 😤😠🤯. Instead use projects + here::here() #rstats

If the first line of your R script is

```
setwd("C:\Users\jenny\path\that\only\I\have")
```

I* will come into your office and
SET YOUR COMPUTER ON FIRE 🔥.

* or maybe Timothée Poisot w

If the first line of your R script is

```
rm(list = ls())
```

I will come into your office and
SET YOUR COMPUTER ON FIRE 🔥.

4:50 PM - 10 Dec 2017

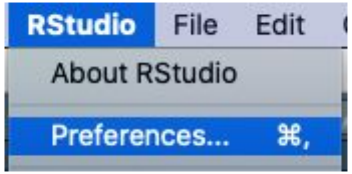
290 Retweets 950 Likes

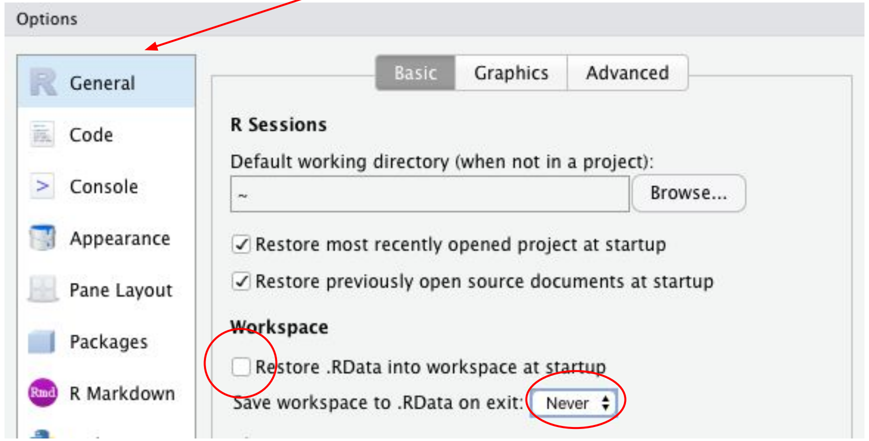
https://www.tidyverse.org/blog/2017/12/workflow-vs-script/

Save your source code, not the workspace

Do not save your .Rdata workspace:

(untick and select 'never')





R Sessions

Default working directory (when not in a project): ~

✓ Restore most recently opened project at startup

✓ Restore previously open source documents at startup

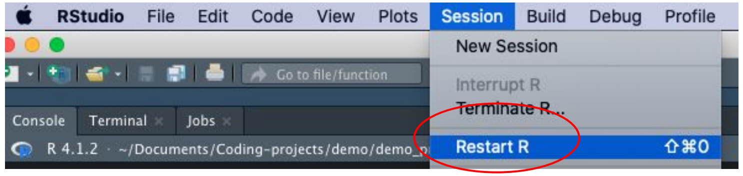
Workspace

☐ Restore .RData into workspace at startup

Save workspace to .RData on exit: Never

Do not use `rm(list = ls())`

Restart R daily* to ensure a clean environment:



Save important objects and intermediates to files and scripts, in a modular way

If there is no source, it's not reproducible

Why setting directory does not work

Don't use `setwd()`

```
setwd("path/that/only/works/on/my/machine")
```

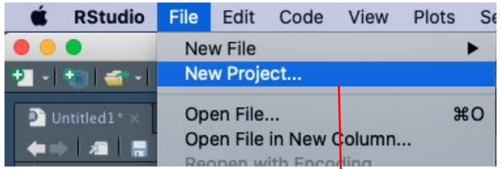
- Only works on your system
- Unlikely to work in a few years or on another computer
- Relying on hard-coded paths make a project easy to break and potentially not reproducible (shared data paths on the server is a different matter)

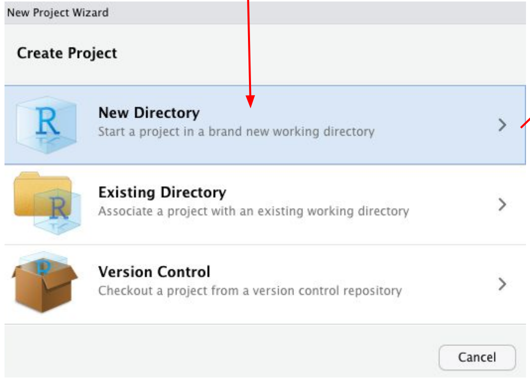
The solution to both:

➢ keep your work as an .Rproj (i.e. project-oriented workflow) to help you manage your workspace env and file paths

Project-oriented workflow

Use Rstudio / .Rproj for your data analysis projects




New Project Wizard

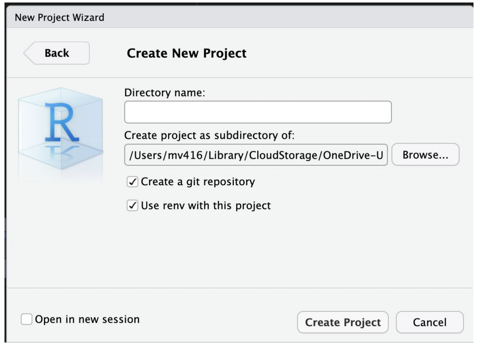
Create Project

New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

Cancel


New Project Wizard

Back **Create New Project**

Directory name:

Create project as subdirectory of: /Users/mv416/Library/CloudStorage/OneDrive-U Browse...

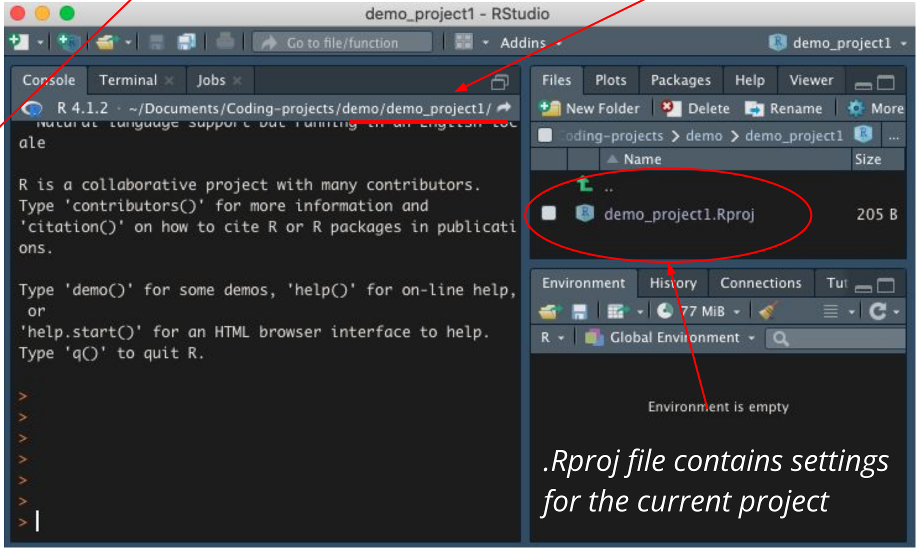
☑ Create a git repository

☑ Use renv with this project

☐ Open in new session

Create Project Cancel

Project directory


demo_project1 - RStudio

```
R 4.1.2 · ~/Documents/Coding-projects/demo/demo_project1/
Natural language support but running in an English loc
ale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publicati
ons.

Type 'demo()' for some demos, 'help()' for on-line help,
 or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
>
>
>
>
>
> |
```

oding-projects > demo > demo_project1

Name	Size
..	
demo_project1.Rproj	205 B

Environment is empty

.Rproj file contains settings for the current project

This means that you are essentially compartmentalizing your current project

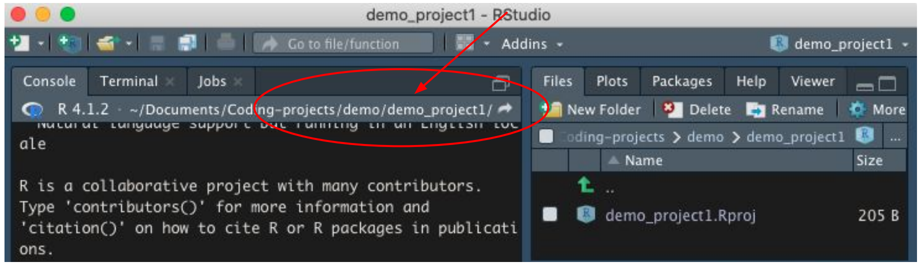
Using .Rproj for your data analysis projects:

File system discipline: project directory stores all your data, scripts, figures

File path discipline: The working directory is set to the project directory (e.g. **demo_project1/**), so you don't need to *setwd()* or specify full paths to data (only internal subfolders that are relative to top directory)

Working directory intentionality: when working on project A, your working directory is set to project A's folder

Project working directory

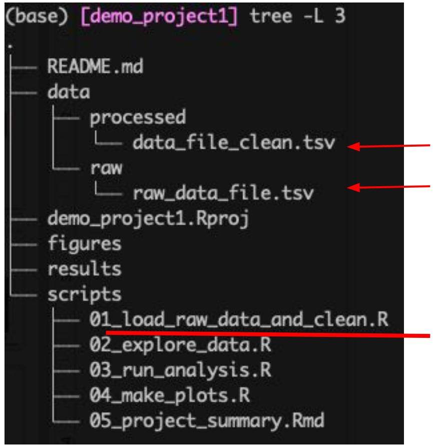


```
Natural language support but running in an English loc
ale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publicati
ons.
```

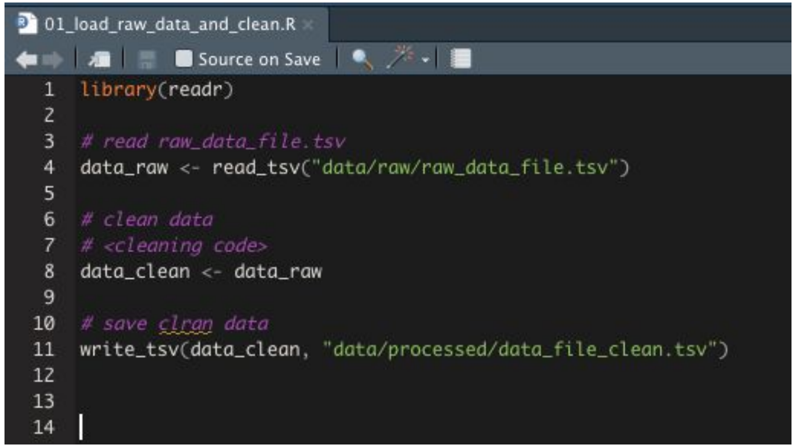
- ➢ The project creates everything it needs, within its workspace/folder, and touches nothing it did not create
- ➢ Any scripts are written assuming they will be run from a fresh R session within the project
- ➢ The project folder can be moved _*anywhere*_, and everything will still work (no paths will be broken)

You don’t need to use setwd()

Keeping your work as an .**Rproj** will help you manage your file paths:



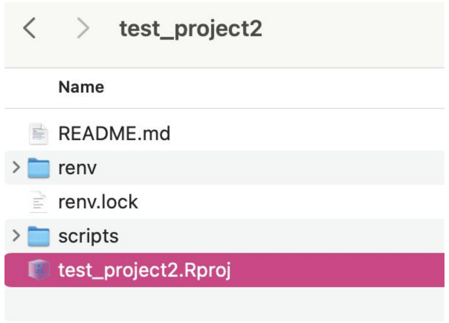
```
(base) [demo_project1] tree -L 3
.
├── README.md
├── data
│   ├── processed
│   │   └── data_file_clean.tsv
│   └── raw
│       └── raw_data_file.tsv
├── demo_project1.Rproj
├── figures
├── results
└── scripts
    ├── 01_load_raw_data_and_clean.R
    ├── 02_explore_data.R
    ├── 03_run_analysis.R
    ├── 04_make_plots.R
    └── 05_project_summary.Rmd
```



```r
library(readr)

# read raw_data_file.tsv
data_raw <- read_tsv("data/raw/raw_data_file.tsv")

# clean data
# <cleaning code>
data_clean <- data_raw

# save clran data
write_tsv(data_clean, "data/processed/data_file_clean.tsv")


```

You don’t need to use rm(list=ls())

Click on .Rproj file in project directory to open a new clean session in your project:

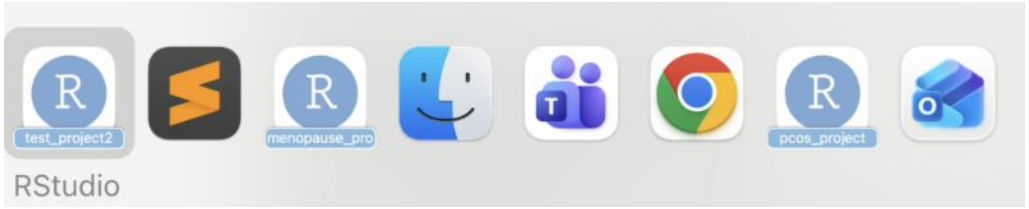


test_project2

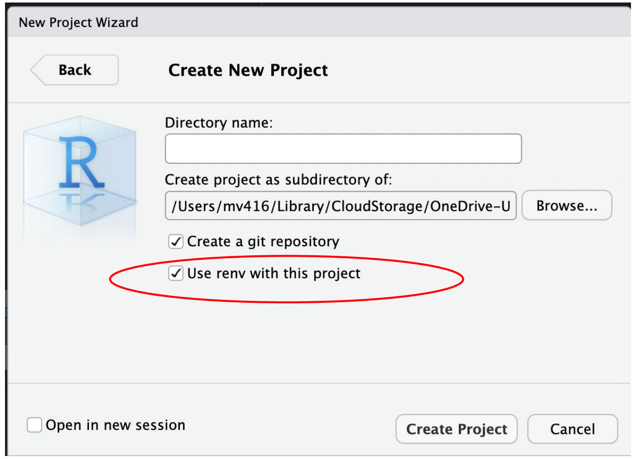
Name

README.md
renv
renv.lock
scripts
test_project2.Rproj

You can also have multiple non-conflicting projects open at the same time:



renv for managing package versions within projects



New Project Wizard

Back Create New Project

Directory name:

Create project as subdirectory of:
/Users/mv416/Library/CloudStorage/OneDrive-U

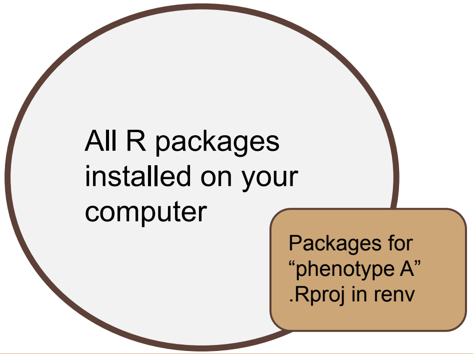
✓ Create a git repository

✓ Use renv with this project

Open in new session

renv package helps you create reproducible environments for your R projects

- Keeps track of all your installed packages
- Installs them in a new location (i.e. easy to share)
- Useful for working on UKB-RAP/AoU platforms (as a part of .Rproj)

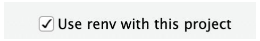

All R packages installed on your computer
Packages for "phenotype A" .Rproj in renv

Setting up your **renv**

`install.packages("renv")`

`renv::init()` – initialise your project env

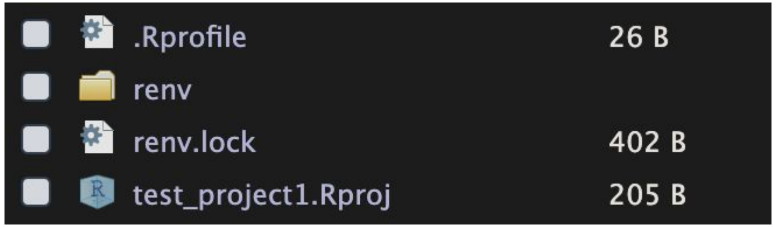
NB this is the same as



☑ Use renv with this project

This creates:

- **renv/**: stores packages for the project.
- **renv.lock:** records packages and the exact versions used
- **.Rprofile**: ensures renv activates when the project opens



```
.Rprofile            26 B
renv
renv.lock           402 B
test_project1.Rproj 205 B
```

When renv is initialised, your project gets its own project library - a private folder (**renv/library/**) where packages are installed just for this project.

This means package versions are isolated and recorded for each project, so updating a package in one project won't break others.

This makes collaboration and reproducibility much easier.

`renv::snapshot()` – update your env with newly installed packages (saves to renv.lock)

`renv::restore()` – when project is re-opened, load all your required packages

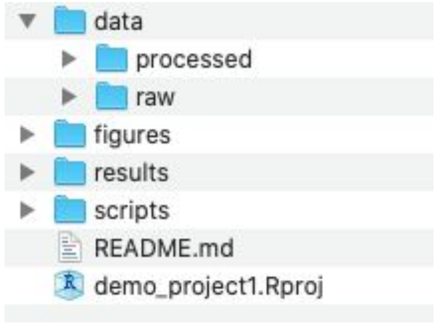
- should not be necessary to run in the local Rstudio, only when cloning a project from Git (eg on UKB-RAP)

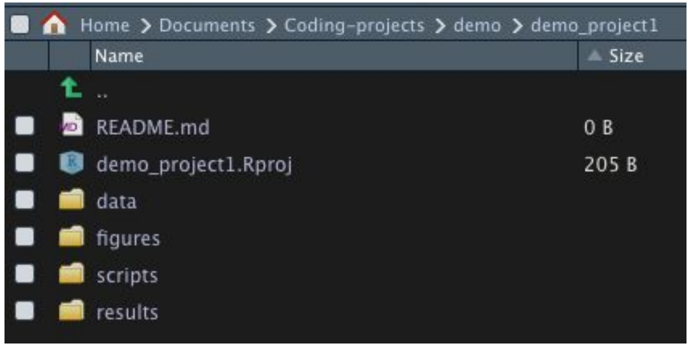
Practical part (15 mins)

1. In your local Rstudio, create a new Project (e.g. *test_project1*) with **renv** and **git** enabled in a new session (window)
    a. Create a new R script in the project folder; install package “sp” in it
    b. Check the “sp” package loads in your project, then *renv::snapshot()* it
    c. Check the “sp” package loads in your other ongoing open Rstudio session
    d. Close your new project Rstudio window
    e. Re-open by clicking on *test_project1.Rproj* file
    f. Test the “sp” package still loads (do not install again!)
2. Download ‘demo_project1.zip’ from the email and uncompress it
    a. Open the project and explore the folders/files
    b. Has the project been created with *renv*?
    c. Review *00_set_up_renv.R*; restore the env and install a package a new package (eg “sp”)

1.1. Hands-on tips for your R projects

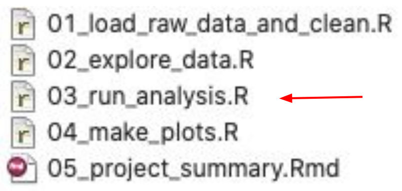
Organise your projects intentionally





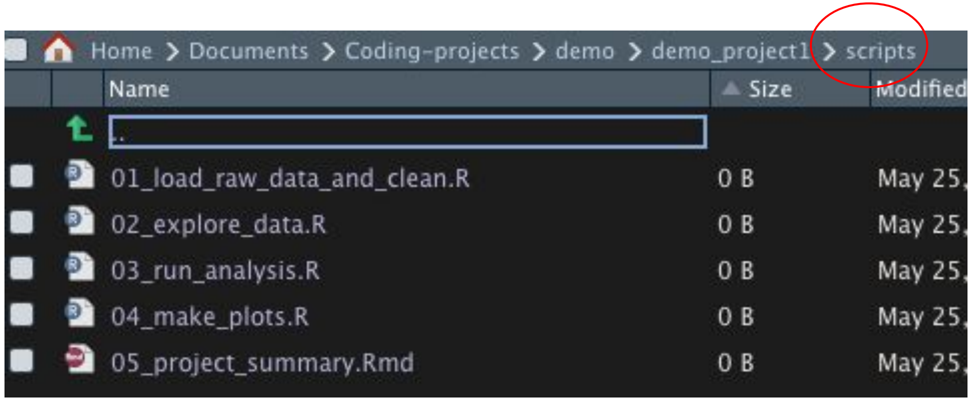
```
README.md             0 B
demo_project1.Rproj   205 B
data
figures
scripts
results
```

Take advantage of default ordering



01_load_raw_data_and_clean.R
02_explore_data.R
03_run_analysis.R
04_make_plots.R
05_project_summary.Rmd

Can have many parts of the analysis separately - save interim results as files and re-read then in the next script

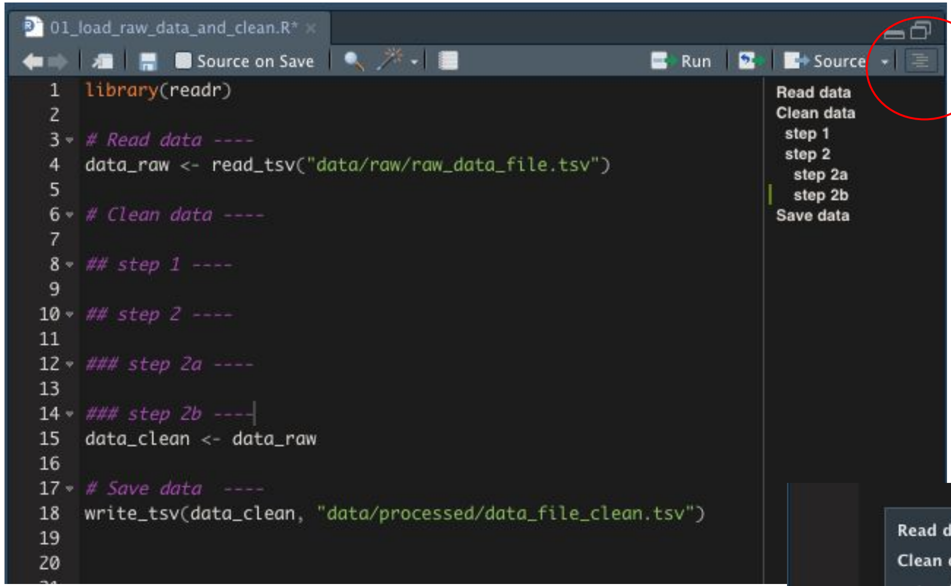


Home > Documents > Coding-projects > demo > demo_project1 > scripts

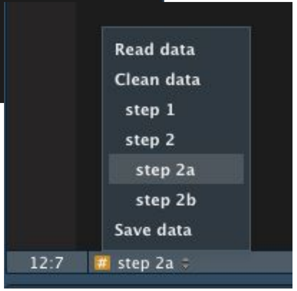
Name	Size	Modified
..		
01_load_raw_data_and_clean.R	0 B	May 25,
02_explore_data.R	0 B	May 25,
03_run_analysis.R	0 B	May 25,
04_make_plots.R	0 B	May 25,
05_project_summary.Rmd	0 B	May 25,

"How to name files" by Jenny Bryan - link (5 mins)

Name your code sections and use them for quick navigation



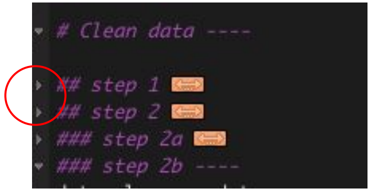
```
library(readr)

# Read data ----
data_raw <- read_tsv("data/raw/raw_data_file.tsv")

# Clean data ----

## step 1 ----

## step 2 ----

### step 2a ----

### step 2b ----
data_clean <- data_raw

# Save data ----
write_tsv(data_clean, "data/processed/data_file_clean.tsv")
```



- Use section headings:

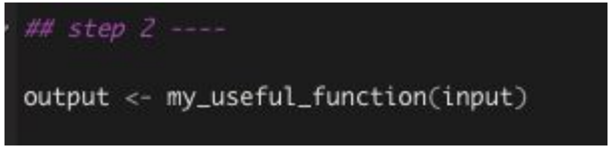
```
# section  ----

## subsection  ----

### subsubsection  ----
```

- Great for navigating in long scripts
- Can fold sections

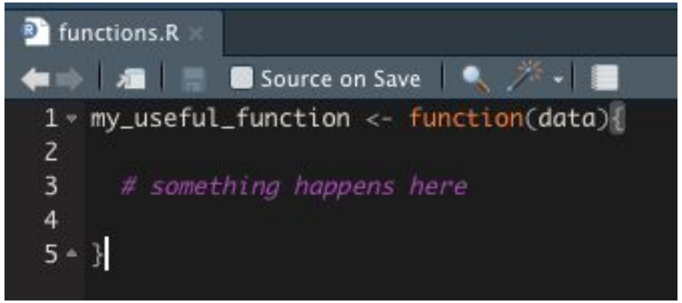


```
# Clean data ----

## step 1
## step 2
### step 2a
### step 2b ----
```

Jump to function definition or open data frame



```
## step 2 ----

output <- my_useful_function(input)
```



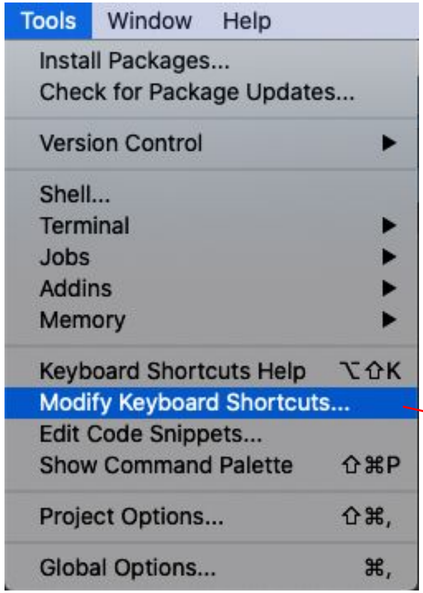
```
my_useful_function <- function(data){

  # something happens here

}
```

To navigate to a function definition or review a function from a package:

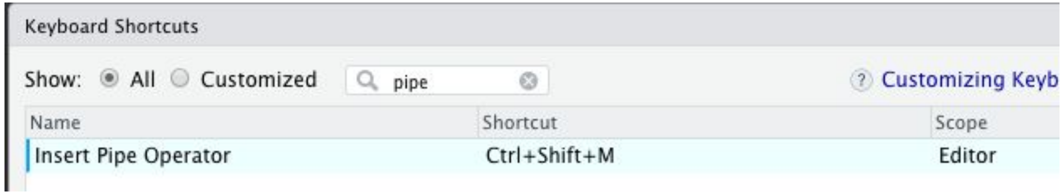
- **Cmd + click** on the function name

> to see what it does internally / check arguments

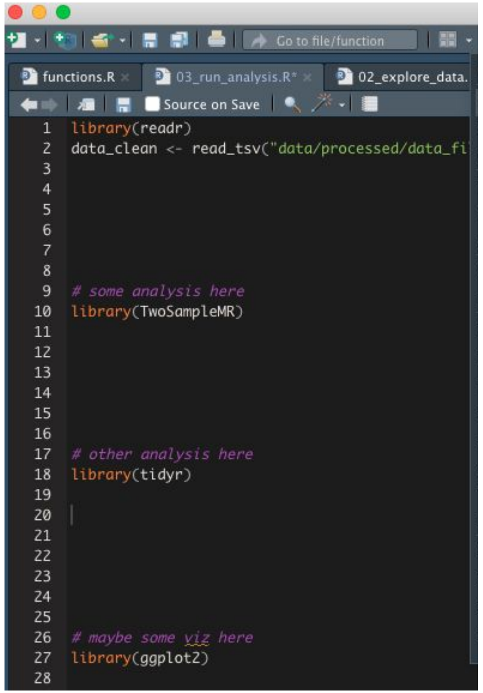
Keyboard shortcuts



- Run (option + Enter)
- `<-` (option/alt + “ - “)
- `%>%` (control + shift + M)
- ```` ```{r} ```` (control + shift + I)

```
```
```


Keyboard Shortcuts

Show: [✓ All] Customized pipe Customizing Keyb

| Name | Shortcut | Scope |
|---|---|---|
| Insert Pipe Operator | Ctrl+Shift+M | Editor |
```

Move all libraries to the top

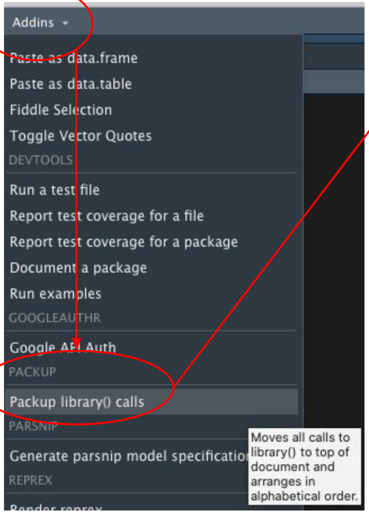


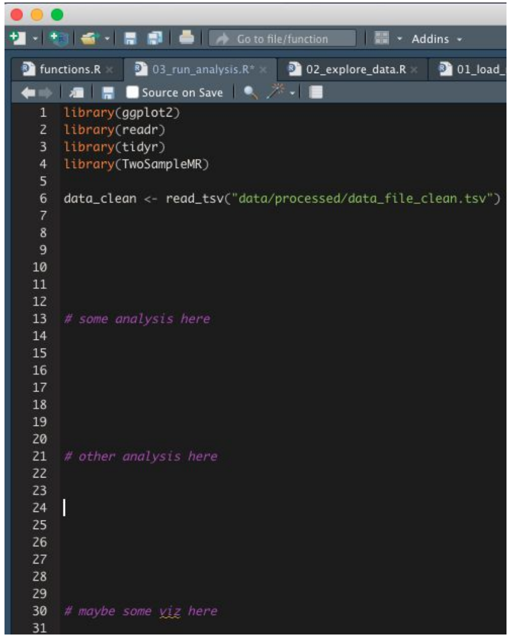
```
library(readr)
data_clean <- read_tsv("data/processed/data_fi



# some analysis here
library(TwoSampleMR)







# other analysis here
library(tidyr)








# maybe some viz here
library(ggplot2)
```

Install *packup* add-in:

```
devtools::install_github("milesmcbain/packup")
```

Call it from Addins menu:



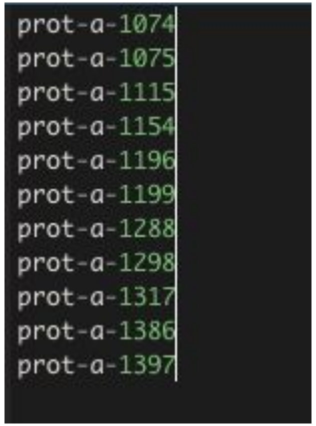


```
library(ggplot2)
library(readr)
library(tidyr)
library(TwoSampleMR)

data_clean <- read_tsv("data/processed/data_file_clean.tsv")






# some analysis here







# other analysis here








# maybe some viz here
```

https://github.com/MilesMcBain/packup

Vertical selection

(hold *option* or *alt* and drag cursor down to select vertically)



```
prot-a-1074
prot-a-1075
prot-a-1115
prot-a-1154
prot-a-1196
prot-a-1199
prot-a-1288
prot-a-1298
prot-a-1317
prot-a-1386
prot-a-1397
```

Great for e.g.

- commenting out a block of code with #
- adding “ “ around a column of ids

Any other R tips to share?

2. Using Git to track changes in your R projects

https://happygitwithr.com/

Checklist before we start

- ➢ GitHub account
- ➢ Personal access token
- ➢ Git is available on your laptop
- ➢ SSH-key (linking Git on your laptop with your GitHub account)

Git brief intro

Git - tool for code version control

- Tracking code changes
- Keeping older versions of the script
- Coding collaboration (with others or yourself in multiple locations)
- Tracking who and when made changes

} If used consistently!

Why use it?

- Can facilitate data/code integrity
- Improves reproducibility (e.g. keeps record of changes)
- Enables code sharing with colleagues / as a part of publication
- Important skill for anyone working with data (e.g. can showcase your work during job applications)

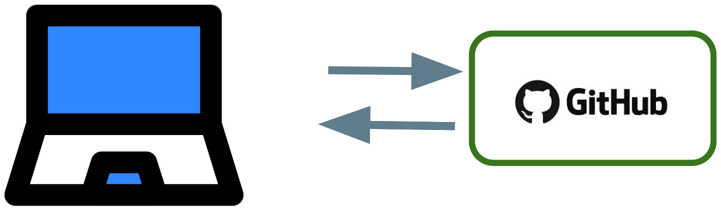
Github / Gitlab etc

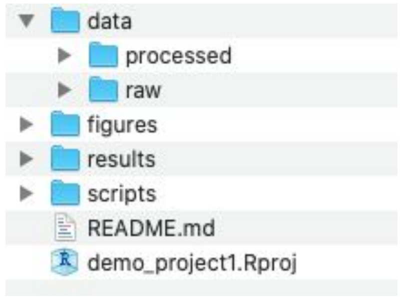
- Platforms where you can store your coding projects using Git

Why might you want to use git? (scenario 1)

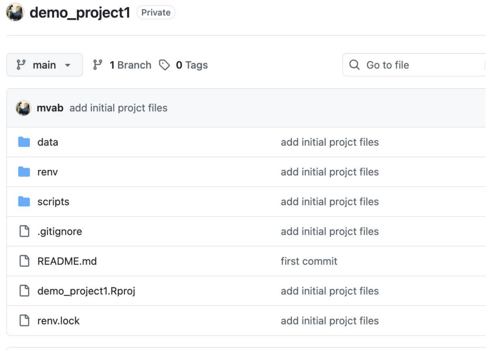
Your R project on local Mac

Your R project as GitHub repo





- data
  - processed
  - raw
- figures
- results
- scripts
- README.md
- demo_project1.Rproj



demo_project1 Private

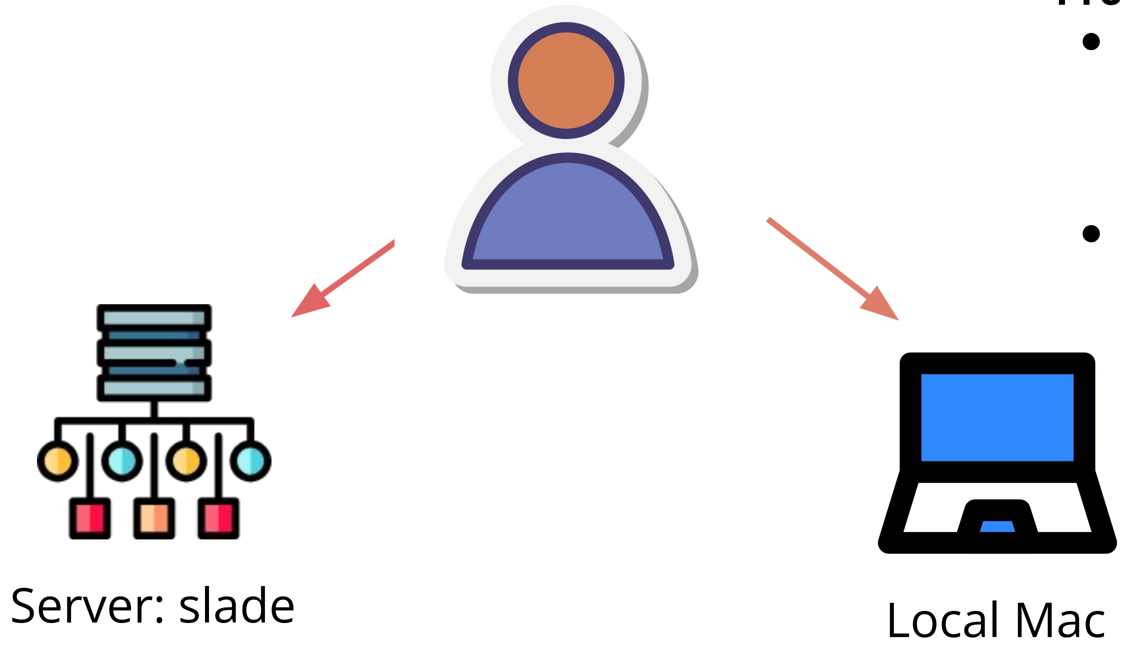
main · 1 Branch · 0 Tags · Go to file

mvab add initial projct files

data	add initial projct files
renv	add initial projct files
scripts	add initial projct files
.gitignore	add initial projct files
README.md	first commit
demo_project1.Rproj	add initial projct files
renv.lock	add initial projct files

- Using git helps you keep track of your daily changes
- All your work is backed-up (as long as you do it regularly and intentionally)
- "Collaborating with yourself" in one place

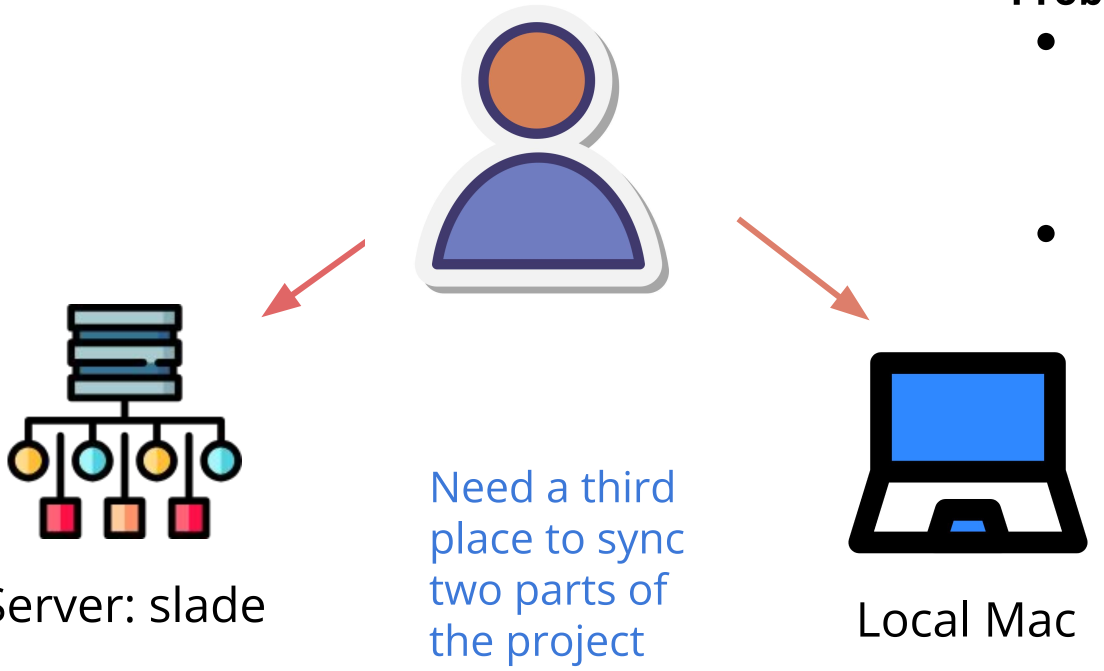
Why might you want to use git? (scenario 2)



Problem:

- The same project is both locations (e.g due to data restrictions / tool availability)
- Don’t want to copy over scripts all the time, as it might get messy!

Why might you want to use git? (scenario 2)

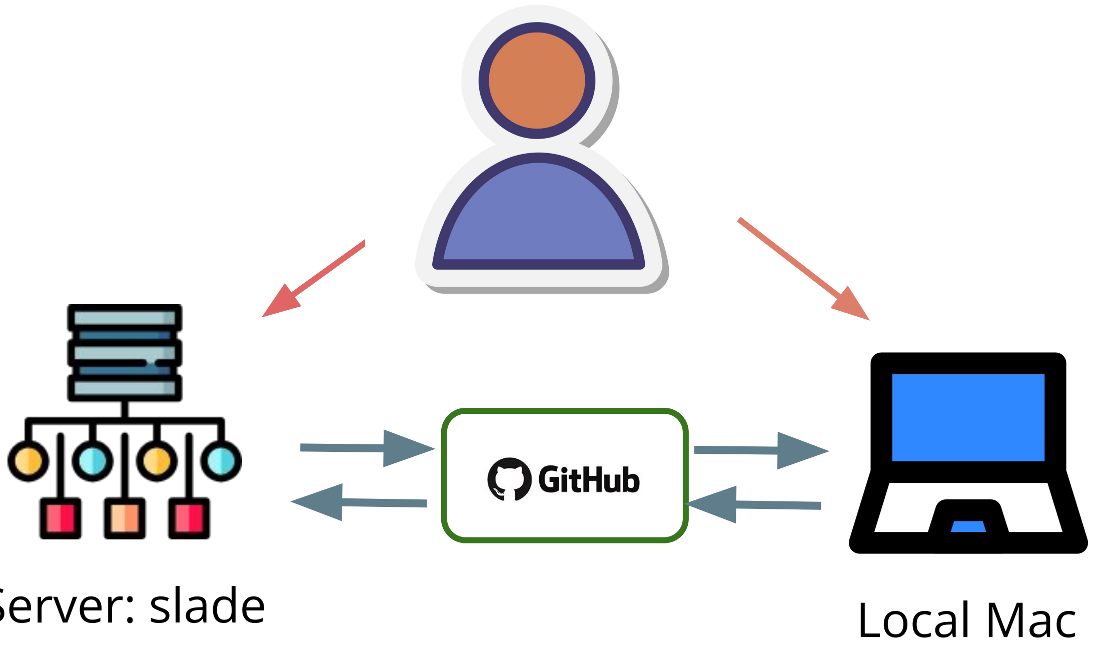


Need a third place to sync two parts of the project

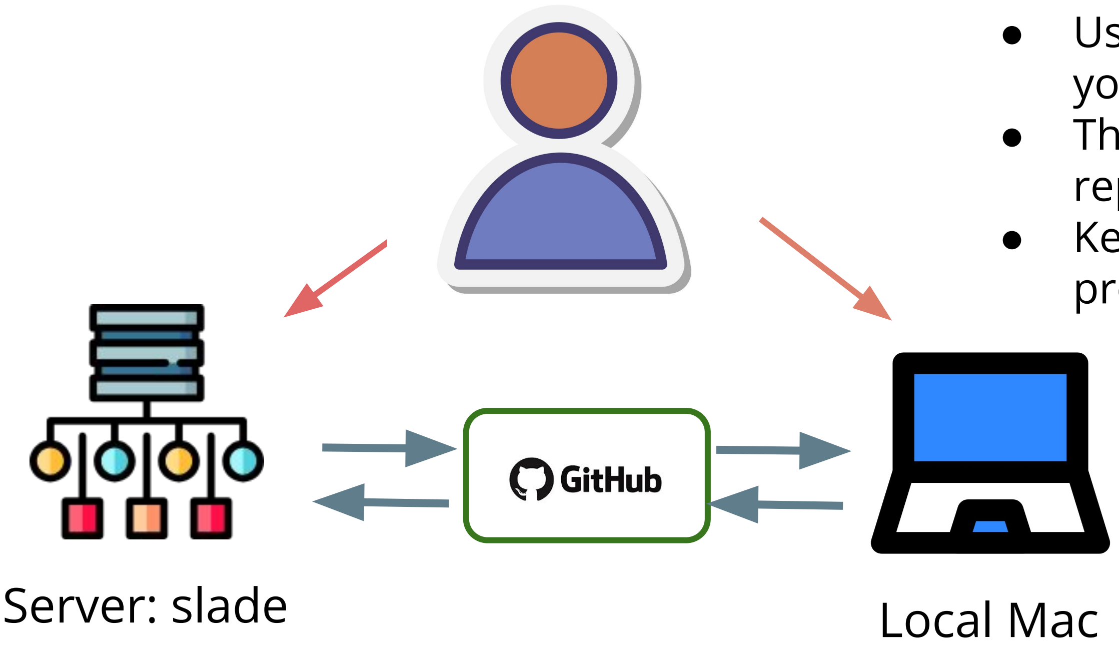
Problem:

- The same project is both locations (e.g due to data restrictions / tool availability)
- Don’t want to copy over scripts all the time, as it might get messy!

Why might you want to use git? (scenario 2)



Why might you want to use git? (scenario 2)



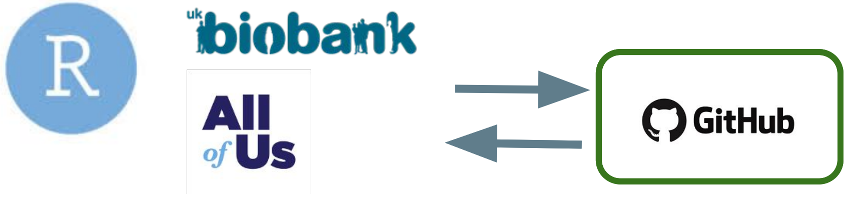
Solution:

- Use git to "collaborate with yourself" in two places
- This enhances project reproducibility
- Keeps all analysis code in one project folder/git repository

Why might you want to use git? (scenario 3)

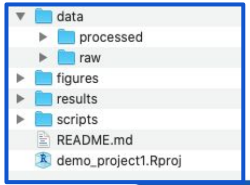
Your R project on

Your R project as GitHub repo



- Using git helps you keep track of your daily changes
- All your work is backed-up (you are ‘forced’ to use Git daily to back up your work, because you start a new R instance every time you work on your project)

Basic git commands / actions



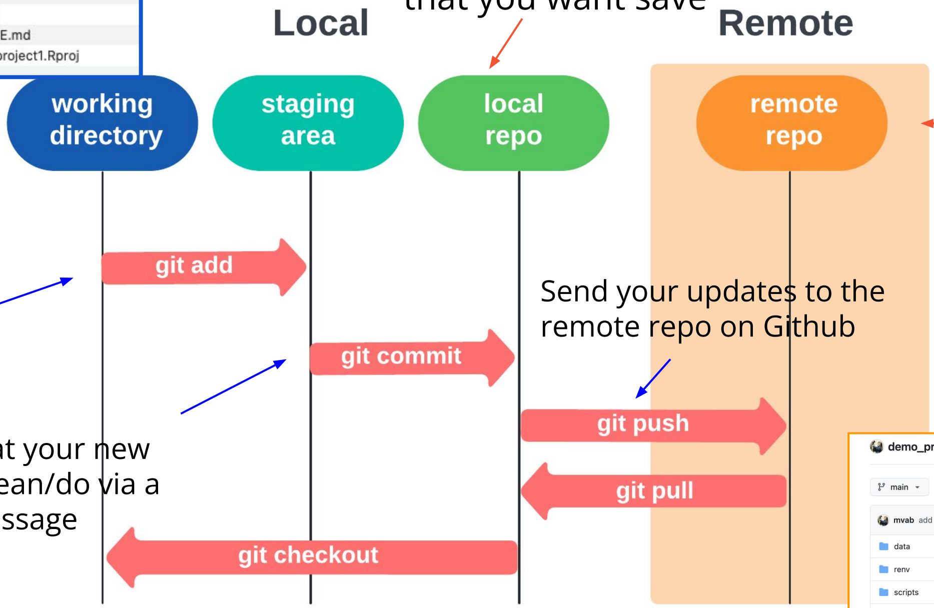
```
▼ data
    ▶ processed
    ▶ raw
▶ figures
▶ results
▶ scripts
  README.md
  demo_project1.Rproj
```

Analysis folder on your local computer

Tell Git which files you changed/created, and want to track with Git

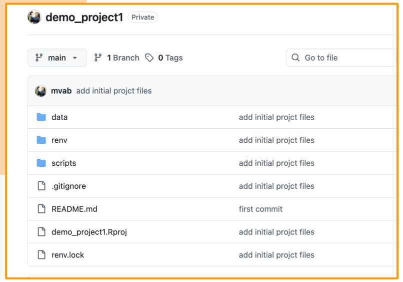
Tell Git what your new changes mean/do via a commit message

Git local repo will keep a record of the new changes that you want save



Send your updates to the remote repo on Github

Your analysis folder repo on Github remote repo with all changes tracked, with a commit message that explains what has changes



demo_project1 Private

main · 1 Branch · 0 Tags · Go to file

mvab add initial projct files

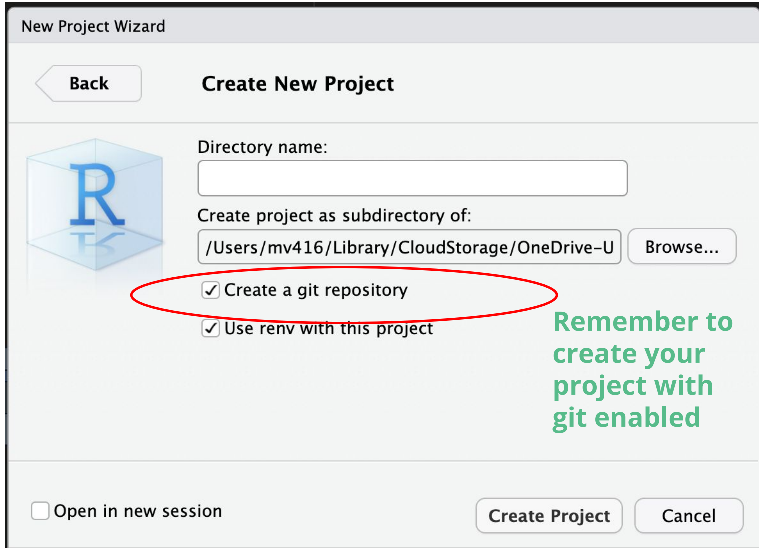
data	add initial projct files
renv	add initial projct files
scripts	add initial projct files
.gitignore	add initial projct files
README.md	first commit
demo_project1.Rproj	add initial projct files
renv.lock	add initial projct files

Follow along if you can

Using git with your .Rproj

- Rstudio user interface makes it easy to get started with git
- (but also can use the terminal to run git commands)

This means Git is enabled in the current project (use drop-down to interact with it)


New Project Wizard

Back Create New Project

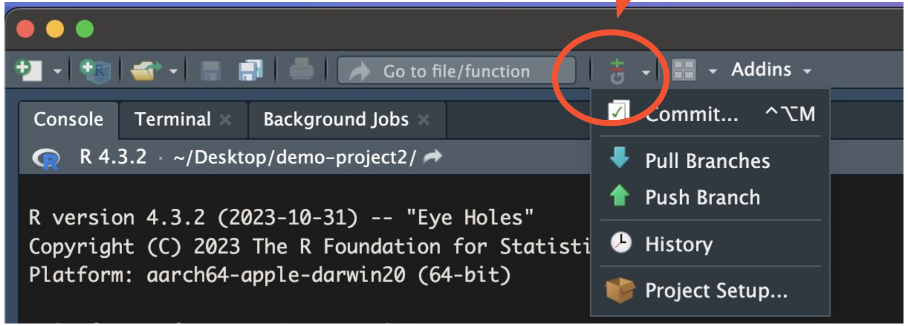
Directory name:

Create project as subdirectory of:
/Users/mv416/Library/CloudStorage/OneDrive-U Browse...

☑ Create a git repository
☑ Use renv with this project

Remember to create your project with git enabled

☐ Open in new session Create Project Cancel



```
R 4.3.2 · ~/Desktop/demo-project2/

R version 4.3.2 (2023-10-31) -- "Eye Holes"
Copyright (C) 2023 The R Foundation for Statisti
Platform: aarch64-apple-darwin20 (64-bit)
```

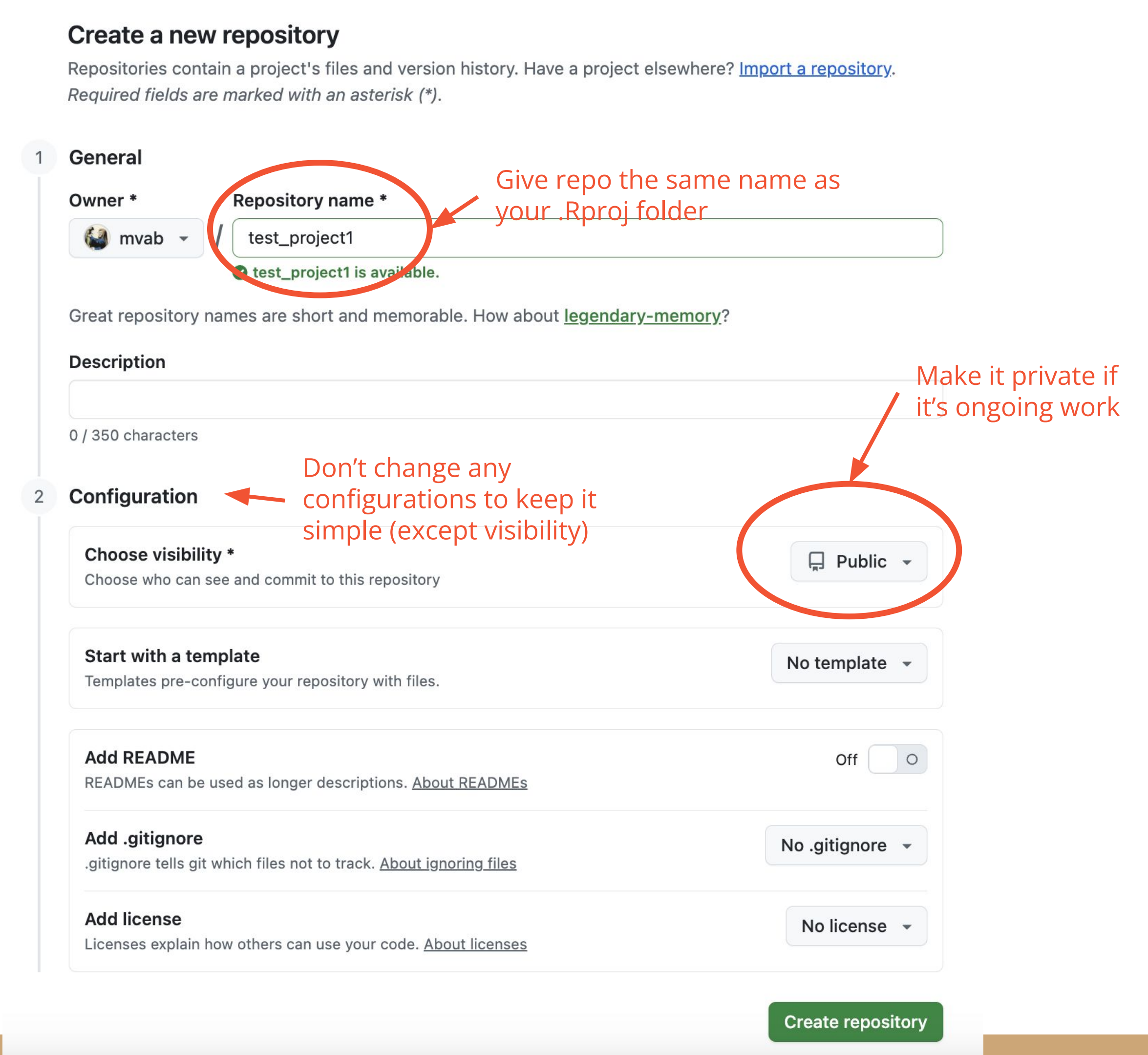
-> Next need to link your local project folder to a GitHub repo

Create remote repo

After logging in your Github account:

To create a new repository:

Repositories -> New



Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

1 **General**

Owner * mvab / **Repository name *** test_project1

test_project1 is available.

Great repository names are short and memorable. How about legendary-memory?

Description

0 / 350 characters

2 **Configuration**

Choose visibility * — Choose who can see and commit to this repository — Public

Start with a template — Templates pre-configure your repository with files. — No template

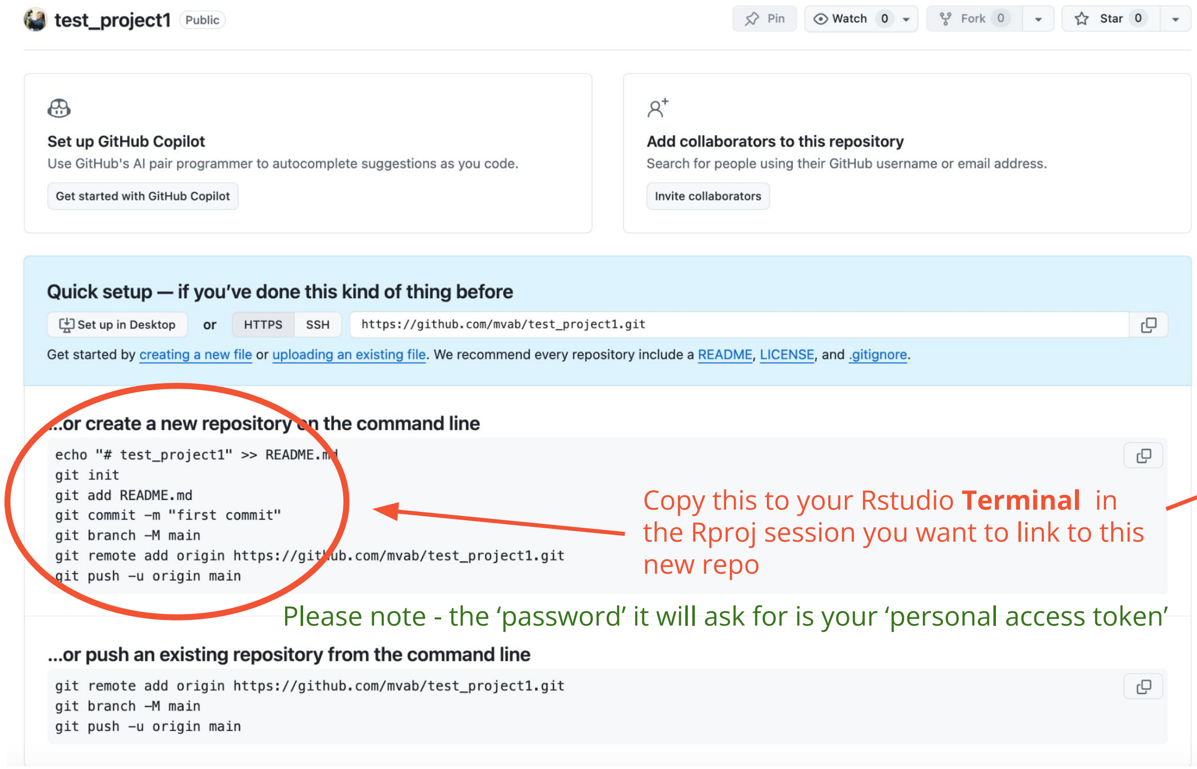
Add README — READMEs can be used as longer descriptions. About READMEs — Off

Add .gitignore — .gitignore tells git which files not to track. About ignoring files — No .gitignore

Add license — Licenses explain how others can use your code. About licenses — No license

Give repo the same name as your .Rproj folder
Make it private if it's ongoing work
Don't change any configurations to keep it simple (except visibility)
Create repository

GitHub makes it easy to link your .Rproj to the new repo:



Set up GitHub Copilot
Use GitHub's AI pair programmer to autocomplete suggestions as you code.

Add collaborators to this repository
Search for people using their GitHub username or email address.

Quick setup — if you've done this kind of thing before

```
https://github.com/mvab/test_project1.git
```

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

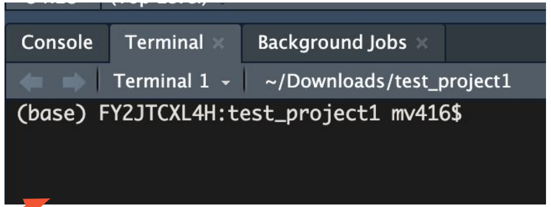
```
echo "# test_project1" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/mvab/test_project1.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/mvab/test_project1.git
git branch -M main
git push -u origin main
```

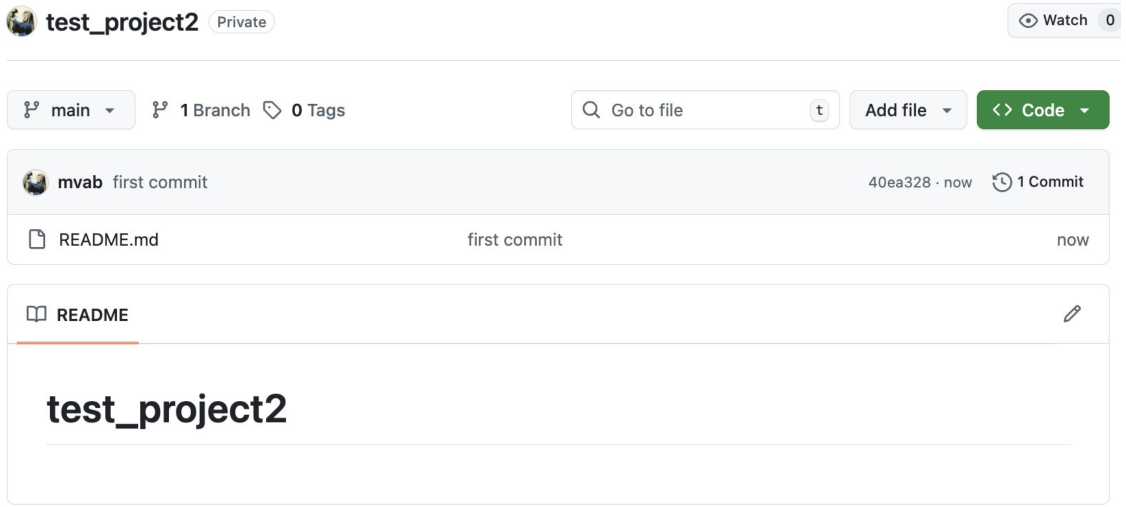
Copy this to your Rstudio **Terminal** in the Rproj session you want to link to this new repo

Please note - the 'password' it will ask for is your 'personal access token'



```
(base) FY2JTCXL4H:test_project1 mv416$
```

Remote repo after first commit


test_project2 Private

main · 1 Branch · 0 Tags

mvab first commit 40ea328 · now 1 Commit

README.md first commit now

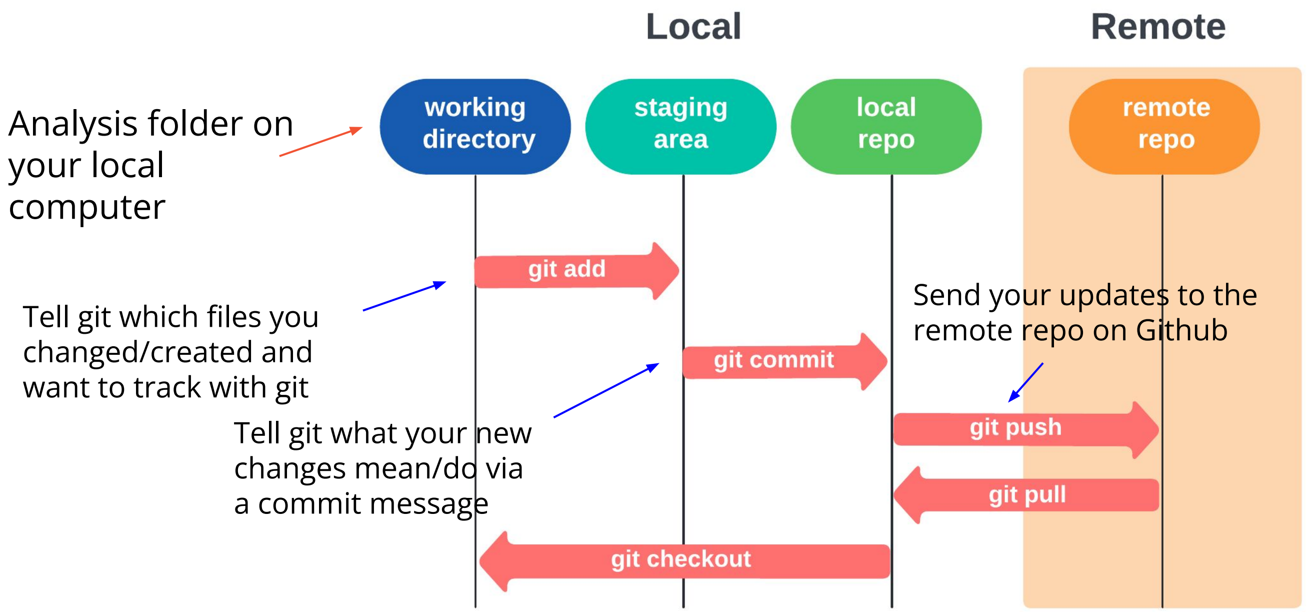
README

test_project2

Only README file has been added

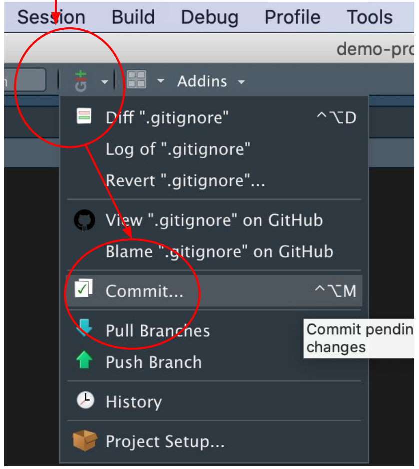
```
echo "# test_project1" >> README.md
git init  <- initialises your project repo
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/mvab/test_project1.git
git push -u origin main
```

initialises your project repo


Local
Remote
Analysis folder on your local computer
working directory
staging area
local repo
remote repo
git add
Tell git which files you changed/created and want to track with git
git commit
Tell git what your new changes mean/do via a commit message
Send your updates to the remote repo on Github
git push
git pull
git checkout

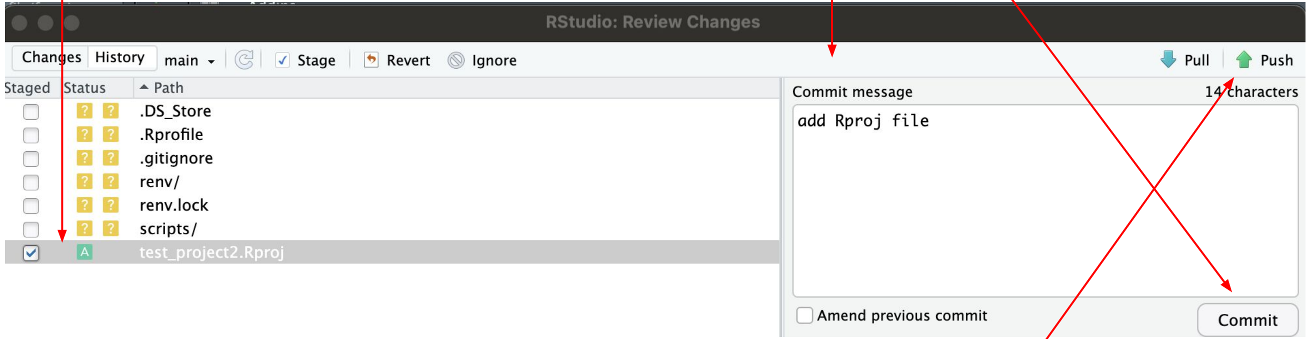
Time to add other files!

1) Git button - > Commit



2) Select the files you want Git to **add** (track):

3) write your 'commit message', ie description of your change (tip: write it as an instruction), then press "**Commit**"

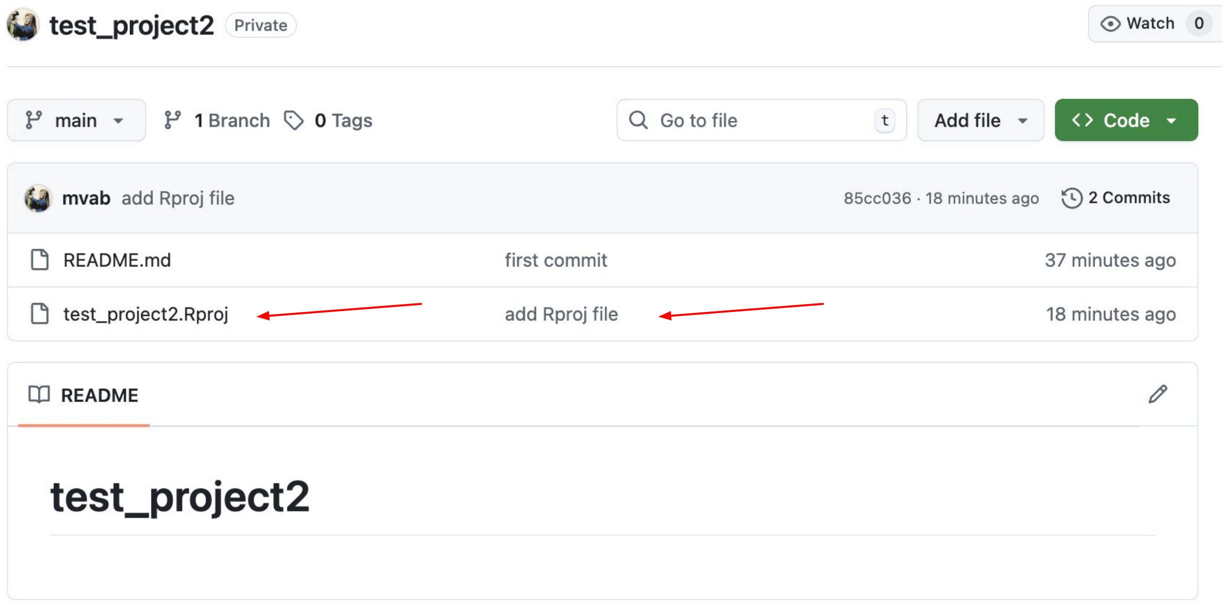


Staged	Status	Path
☐	? ?	.DS_Store
☐	? ?	.Rprofile
☐	? ?	.gitignore
☐	? ?	renv/
☐	? ?	renv.lock
☐	? ?	scripts/
✓	A	test_project2.Rproj

```
add Rproj file
```

4) Press "**Push**" to send the change to remote repo

Your change has been added:


test_project2 Private

Watch 0

main 1 Branch 0 Tags Go to file t Add file Code

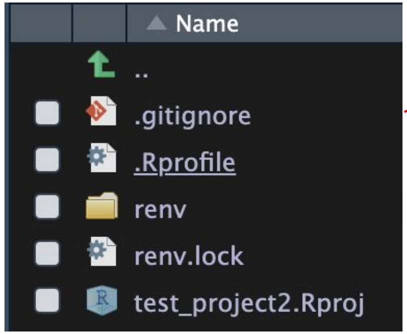
mvab add Rproj file 85cc036 · 18 minutes ago 2 Commits

README.md	first commit	37 minutes ago
test_project2.Rproj	add Rproj file	18 minutes ago

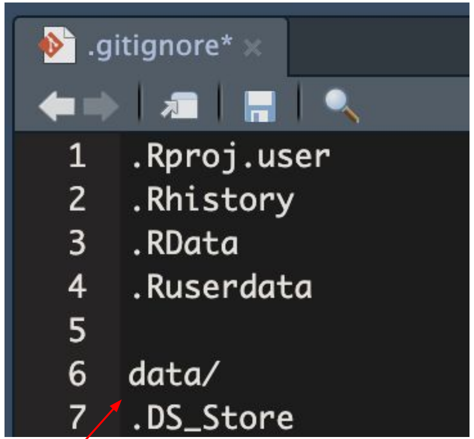
README

test_project2

.gitignore file - list files you don’t want to store in remote repo



.gitignore
.Rprofile
renv
renv.lock
test_project2.Rproj



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

data/
.DS_Store
```

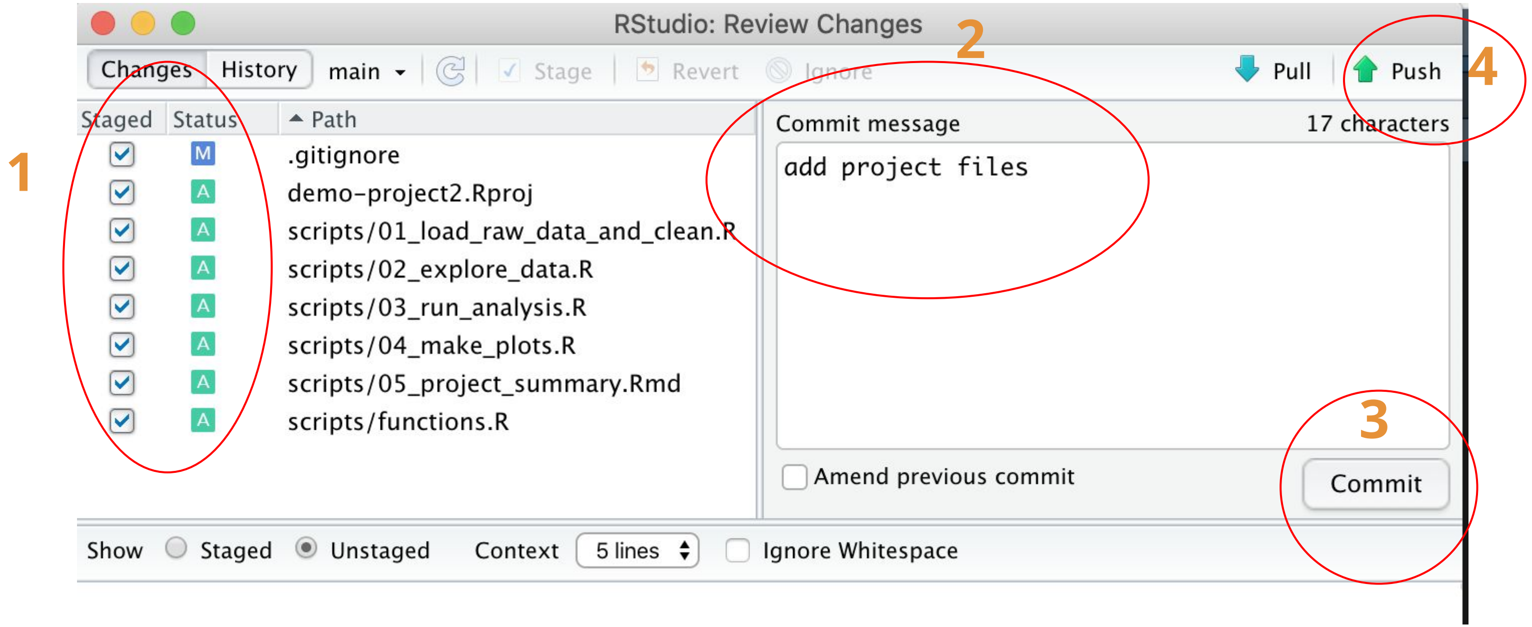
Add **data/** folder to *.gitignore* file so that your data files (if large or sensitive) are not committed to your project repo on Github

- list files and folders in your project folder in the *.gitignore* file - they will be ignore by Git (this prevents them from being accidentally added to GitHub repo)
- save and ‘**add**’ file
- **commit** and **push** the updated *.gitignore* file to your repo

Now add more files:

Commit changes:

Add message:



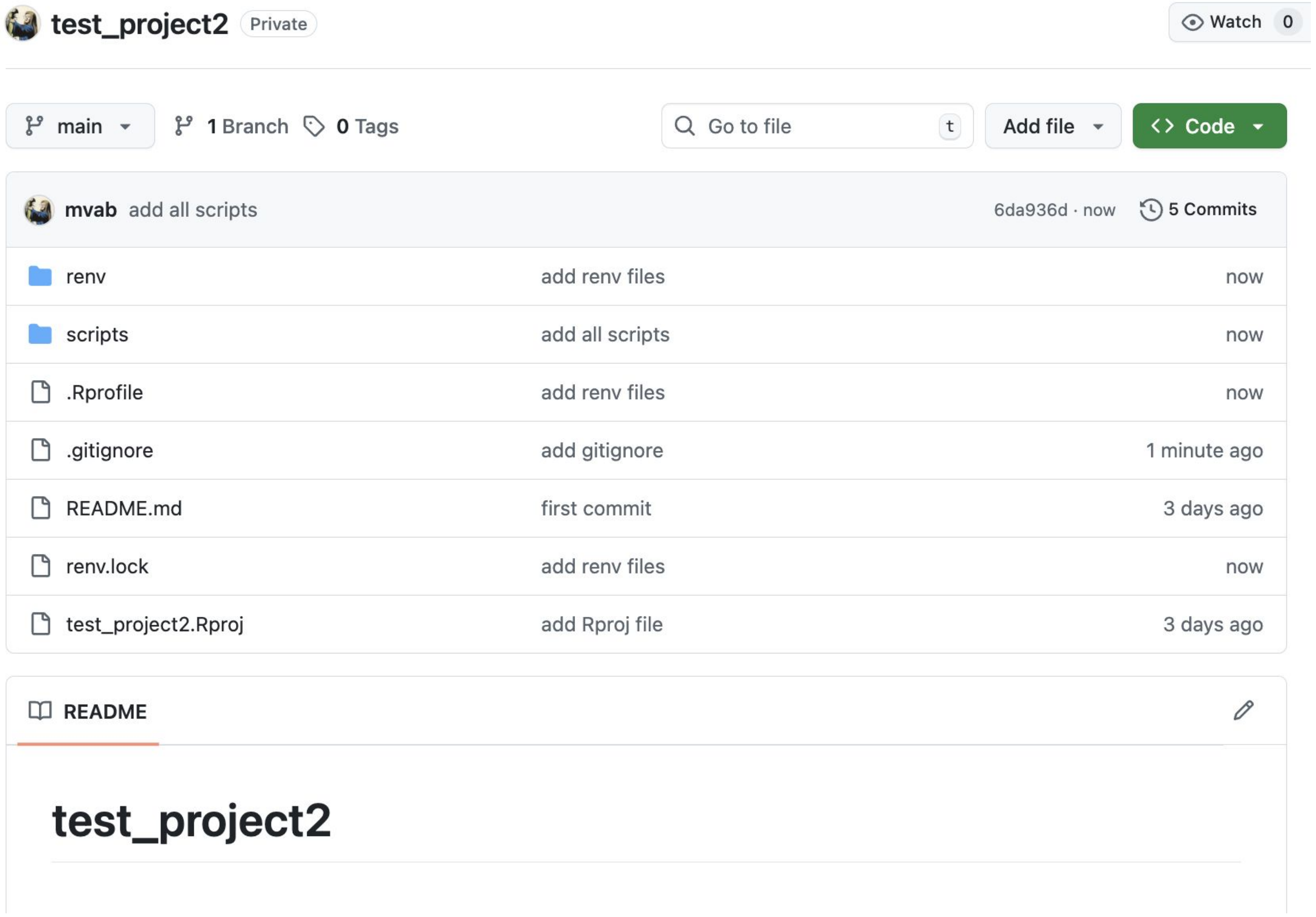
Staged	Status	Path
✓	M	.gitignore
✓	A	demo-project2.Rproj
✓	A	scripts/01_load_raw_data_and_clean.R
✓	A	scripts/02_explore_data.R
✓	A	scripts/03_run_analysis.R
✓	A	scripts/04_make_plots.R
✓	A	scripts/05_project_summary.Rmd
✓	A	scripts/functions.R

Commit message 17 characters

```
add project files
```

Show: [✓ Unstaged] Context: 5 lines

Your changes
on Github:



test_project2 Private

Watch 0

main · 1 Branch · 0 Tags · Go to file · Add file · Code

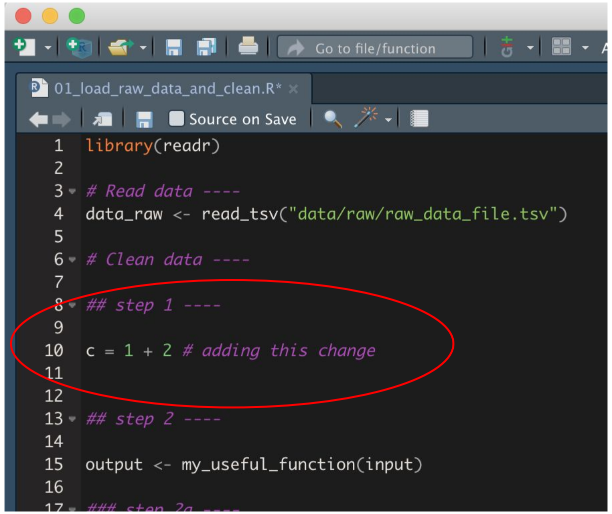
mvab add all scripts · 6da936d · now · 5 Commits

renv	add renv files	now
scripts	add all scripts	now
.Rprofile	add renv files	now
.gitignore	add gitignore	1 minute ago
README.md	first commit	3 days ago
renv.lock	add renv files	now
test_project2.Rproj	add Rproj file	3 days ago

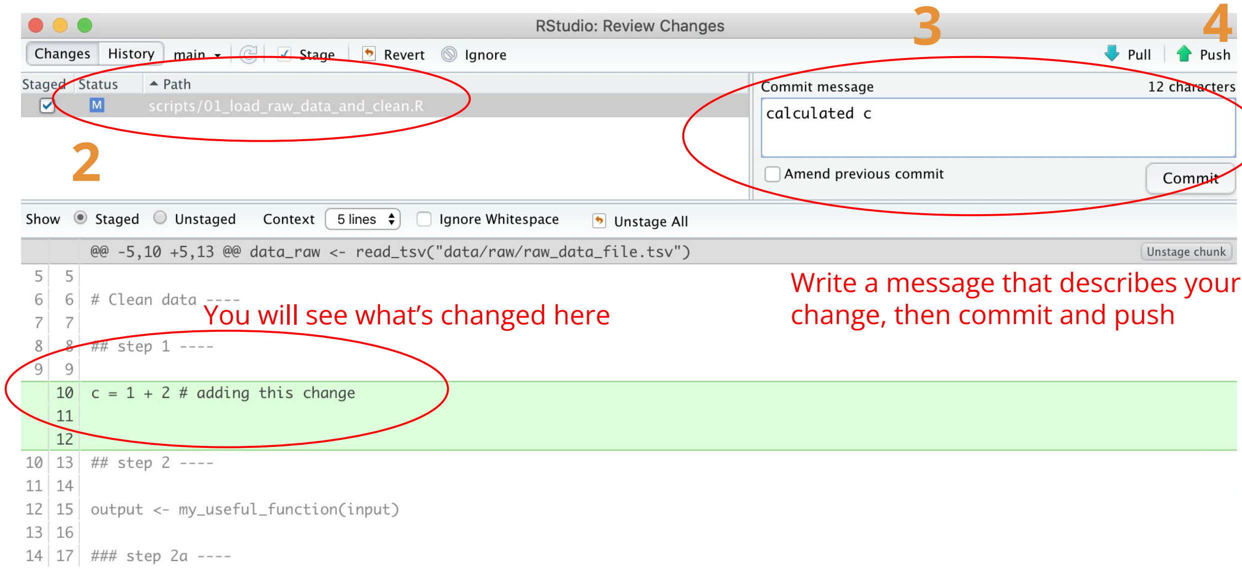
README

test_project2

Practise adding a specific change to a script:



```
library(readr)

# Read data ----
data_raw <- read_tsv("data/raw/raw_data_file.tsv")

# Clean data ----

## step 1 ----

c = 1 + 2 # adding this change


## step 2 ----

output <- my_useful_function(input)

### step 2a ----
```



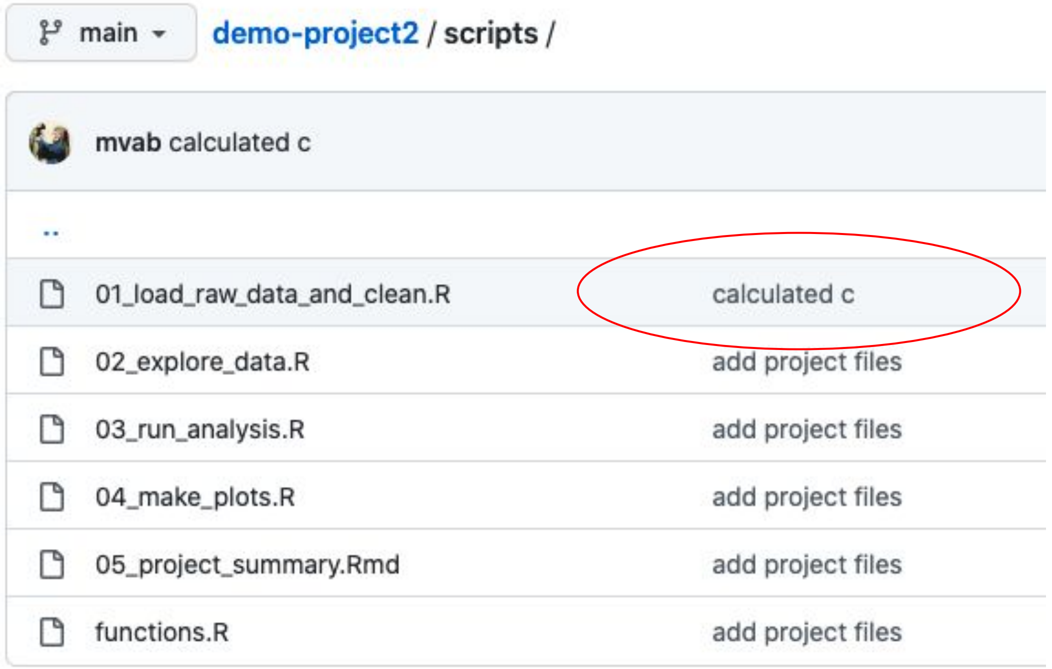
scripts/01_load_raw_data_and_clean.R

Commit message: calculated c

```
@@ -5,10 +5,13 @@ data_raw <- read_tsv("data/raw/raw_data_file.tsv")

# Clean data ----

## step 1 ----

c = 1 + 2 # adding this change


## step 2 ----

output <- my_useful_function(input)

### step 2a ----
```

You will see what's changed here

Write a message that describes your change, then commit and push



demo-project2 / scripts /

mvab calculated c

..

01_load_raw_data_and_clean.R	calculated c
02_explore_data.R	add project files
03_run_analysis.R	add project files
04_make_plots.R	add project files
05_project_summary.Rmd	add project files
functions.R	add project files

Good practice principles

1. **Add** and **commit** your changes often and in small chunks
   - E.g. separate commits for different files and sections in your analysis steps
2. Write descriptive commit messages of what you’ve changed
   - “fix typos” is ok, but “update” is not
3. Use .gitignore
4. Keep a project README
   - describes what your scripts do, and where external data is stored

3. R projects on UKB-RAP/All of Us platforms with Git

Start two new Rstudio instances now!

Working in Rstudio in on-demand instances

The problem:

- You are forced to scratch from a clean env every time you work a project:
  - All packages need to be installed every time
  - All intermediate data tables need to be regenerated
  - To save progress, you have to copy your R/Rmd scripts back and forth with your User project

The solution:

- Organise your R code as *Rproj* with *renv*, and use *Git* to back it up:
  - *renv::restore()* brings back all your packages
  - Intentional and modular file organisation helps you go back your intermediate files quicker
  - Commiting your ongoing changes to your project repo on GitHub keeps your work backed up
  - *git clone* brings back your entire project to your clean Rstudio instance

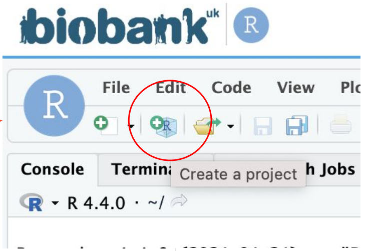
Follow along if you can

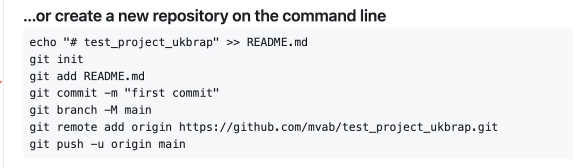
You already know how to make it work

Start a new project in Rstudio instance
(instance #1 you started):

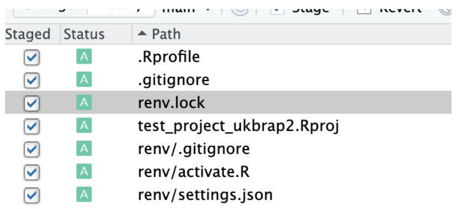
1) Create a new project (with renv + git)
2) On Github, create a new private repo
3) On Rstudio Terminal, authenticate:
   - *git config --global user.email "you@example.com"*
   - *git config --global user.name "user"*
4) Run the set up commands from GitHub on Rstudio Terminal
   - use your token as password!
5) Push all other files to your remote repo

6) Upload or create *00_renv_setup.R* file project folder (attached in the email)
7) Add any other dummy R files you want



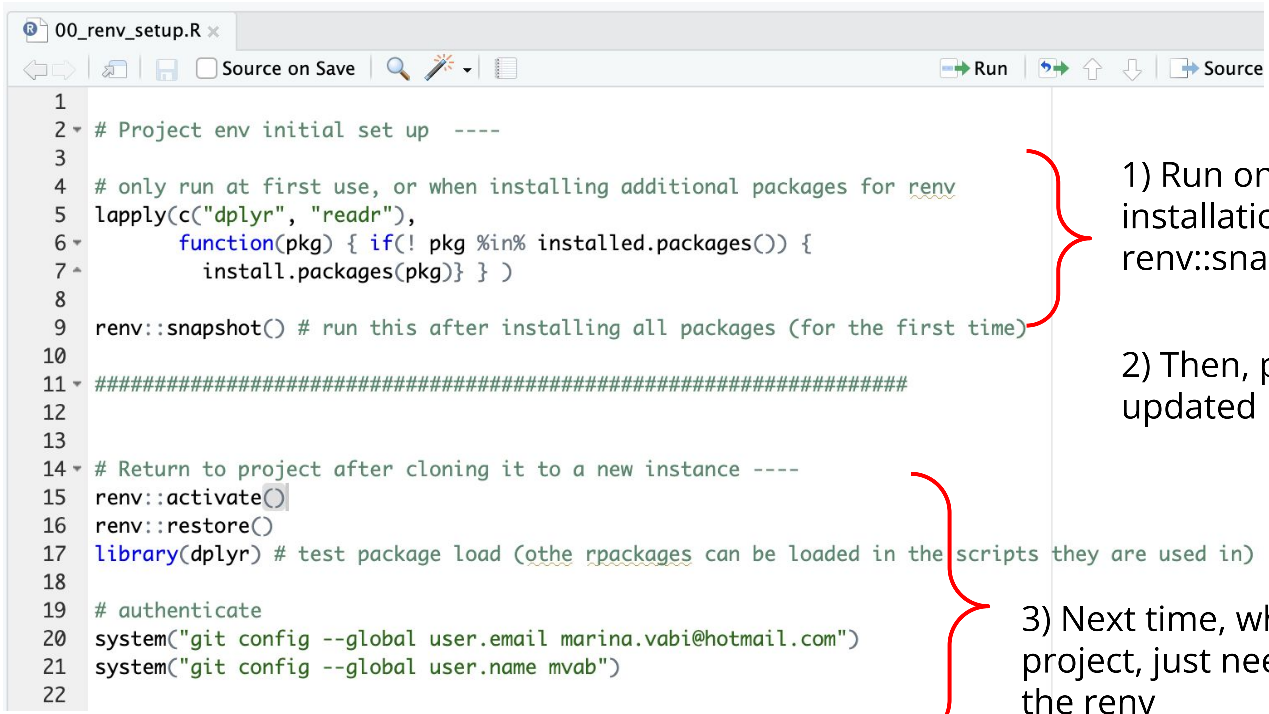

...or create a new repository on the command line

```
echo "# test_project_ukbrap" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/mvab/test_project_ukbrap.git
git push -u origin main
```



Staged	Status	Path
✓	A	.Rprofile
✓	A	.gitignore
✓	A	renv.lock
✓	A	test_project_ukbrap2.Rproj
✓	A	renv/.gitignore
✓	A	renv/activate.R
✓	A	renv/settings.json

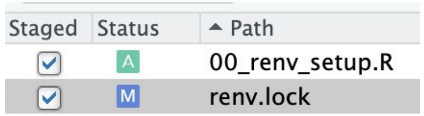
renv setup



```
# Project env initial set up  ----

# only run at first use, or when installing additional packages for renv
lapply(c("dplyr", "readr"),
       function(pkg) { if(! pkg %in% installed.packages()) {
         install.packages(pkg)} } )

renv::snapshot() # run this after installing all packages (for the first time)

####################################################################


# Return to project after cloning it to a new instance ----
renv::activate()
renv::restore()
library(dplyr) # test package load (othe rpackages can be loaded in the scripts they are used in)

# authenticate
system("git config --global user.email marina.vabi@hotmail.com")
system("git config --global user.name mvab")
```

1) Run once, only on the first package installation only + renv::snapshot() to save it to renv.lock

2) Then, push the new file and the updated renv.lock to GitHub:



Staged	Status	Path
✓	A	00_renv_setup.R
✓	M	renv.lock

3) Next time, when going back to this project, just need to activate and restore the renv

Tip: keep a copy of 00_renv_setup.R file in all projects

Ending your R instance working on Rproj

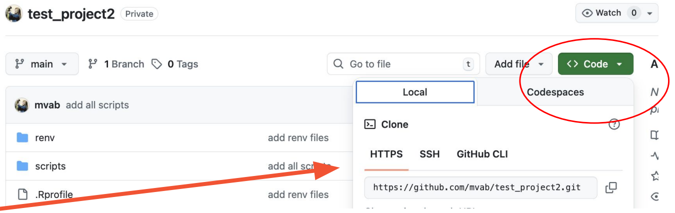
At the end of your session, save and push all your changes to GitHub (check it's all there)

- Remember to run *renv::snapshot()* - so that the *renv.lock* file is updated (push it to GitHub too!)
- Be especially careful if working with individual level data (tip: only save your data files in to folder *data/*, and list that folder in your *.gitignore* - so it does not get added to GitHub by accident)
- Once all code you wrote in the current session is on GitHub, it's safe to terminate instance #1

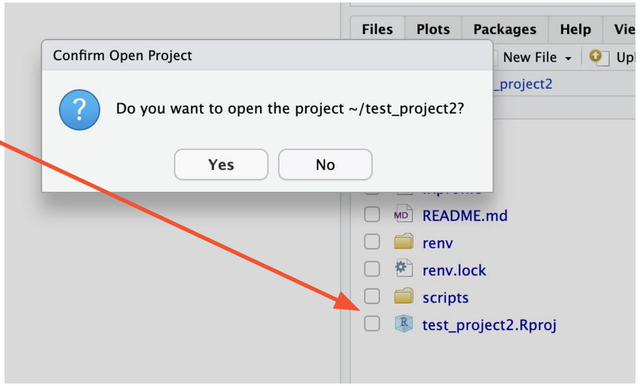
Returning to your R project

1) Starting from a clean instance (instance #2 you started)
2) On Rstudio terminal, *git clone* your project
   - *git clone https://github.com/mvab/test_project2.git*
   - Authenticate with your token
3) Important: click on Rproj file in the project folder to activate the project
4) Open *00_renv_setup.R* file and test that *library(dplyr)* currently does not work!
5) In *00_renv_setup.R*run:
   - *renv::activate()*
   - *renv::restore()*
   - Test dplyr again!

Any other packages you install - remember to run *renv::snapshot()* - so that the renv.lock file is updated (and push to your GitHub repo)



```
https://github.com/mvab/test_project2.git
```



Do you want to open the project ~/test_project2?

Summary: using .Rproj for organising work

- "Reproducible project in R" means:
  - **File system discipline:** all files related to a single project are stored in a designated folder;
  - **Working directory discipline**: intentionally work in project directory when opening Rproj
  - **File path discipline:** paths are relative to the project directory (not hard-coded full paths!)
  - **Daily work habit:** Restarting R often and re-running your script under development from the top will help you catch issues early on
  - **Using git for version control**

- Practising these habits together will give you the biggest pay-off
  - Reproducing your analyses will be easy
  - Organising your projects will help you make sense of them in 6/12/etc months
  - Can move your project anywhere or share it with anyone without changing paths

https://rstats.wtf/project-oriented-workflow.html#work-in-a-project

Final thoughts / disclaimers

- Project-oriented workflow in R is not suitable/applicable to every scenario
    - Sometimes data is stored externally, and can't moved/copied (so you can't use within-project paths, but potentially can use symbolic links)
- Not all work is done interactively in Rstudio
    - Some people use R from the terminal on the server - again, because of data access/size
    - Some analyses are computation-heavy and require to be submitted as scripts / run in parallel on server

- If your current workflow with `setwd()` works for you and your colleagues, that's totally fine, but consider future-proofing! ;)

Recommended and used resources

https://www.tidyverse.org/blog/2017/12/workflow-vs-script/

https://richpauloo.github.io/2018-10-17-How-to-keep-your-R-projects-organized/

https://www.rforecology.com/post/organizing-your-r-studio-projects/

https://kkulma.github.io/2018-03-18-Prime-Hints-for-Running-a-data-project-in-R/

https://rstats.wtf/project-oriented-workflow.html

https://appsilon.com/rstudio-shortcuts-and-tips/

https://datacornering.com/my-favorite-rstudio-tips-and-tricks/

https://happygitwithr.com/

https://rstudio.github.io/renv/articles/renv.html

https://rstats.wtf/

Thank you!