

Version Control and Collaboration with Git and GitHub

Resul Umit

March 2021

[Skip intro — To the contents slide.](#)

[I can teach this workshop at your institution — Email me.](#)

Who am I?

Resul Umit

- post-doctoral researcher at the University of Oslo
- interested in representation, elections, and parliaments
 - **a recent publication**: Parliamentary communication allowances do not increase electoral turnout or incumbents' vote share
- teaching workshops also on
 - **writing reproducible research papers**
 - **working with Twitter data**
 - **creating professional websites**
 - **automated web scraping**
- more information available at **resulmit.com**

The Workshop — Overview

- Half a day, on how to
 - version our academic work
 - store the different versions securely — in multiple places
 - collaborate with co-authors — in writing as well as versioning and storing
- Using Git and GitHub, through
 - command line
 - GitHub Desktop, RStudio
- Most useful for those working with plain text
 - e.g., Markdown, rather than Microsoft Word
 - but not exclusively so
 - projects based on binary files would also benefit
 - you may switch to plain text in the future

The Workshop — Motivation

- Research projects have many versions, as they are typically completed
 - over a long period of time
 - in numerous sittings
 - by multiple authors
 - on multiple computers
- With many versions created over time, in different sittings ..., there emerges various challenges
 - keeping track of changes and versions
 - reverting to a previous version when necessary
 - storing versions safely in multiple locations
 - communicating the versions to other authors and/or computers
 - working on the same project with co-authors at the same time

The Workshop — Workflow

The Git-and-GitHub workflow has two steps

- Git creates, saves, and tracks versions
 - on our own machine
 - but also can also save them elsewhere, such as on GitHub
- Github keeps a copy online
 - allowing for safe storage in the cloud
 - but also for collaboration, through a common version

The Workshop — Workflow — Comparison

- The Git-and-GitHub workflow might seem daunting to adopt at first
 - originally designed for software development
 - here, adopted to academic work
 - but offers many benefits over alternatives
 - efficient, free
- Existing workflows might seem easier to keep
 - we all version control and collaborate, one way or another
 - e.g., renaming, saving, and emailing to co-authors
 - using Dropbox, Google Docs, Overleaf
 - but less efficient in the long run
- Give the Git-and-GitHub workflow a chance
 - if you are curious enough to be here on this slide, this is probably the right workflow for you

The Workshop — Contents

Part 1. Introduction

- e.g., getting ready to work together

Part 2. Tools

- e.g., downloading workshop material

Part 3. Version Control

- .e.g, versioning by typing commands

Part 4. What to Version

- e.g., ignoring

Part 5. Collaboration

- e.g., working on the same files

Part 6. Third-Party Applications

- e.g., using GitHub Desktop

The Workshop — Organisation

- ~~Sit in groups of two~~
 - participants learn as much from their partner as from instructors
 - play the role of co-authors
- Type, rather than copy and paste, the code that you will find on these slides
 - typing is a part of the learning process
 - keyboard shortcuts for copy/paste does not work in the shell
- When you have a question
 - ~~ask your partner~~
 - ~~google together~~
 - if it cannot wait, turn your microphone on and ask
 - if it can wait, type it in the chat

The Workshop — Organisation — Slides*

01:00

Slides with this background colour indicate that your action is required, for

- setting the workshop up
 - e.g., downloading course material
- completing the exercises
 - e.g., saving a different version of a file
 - there are 30+ exercises
 - these slides have countdown timers

* These slides are and will remain available at https://resulunit.com/teaching/git_workshop.html.

The Workshop — Organisation — Slides

- Codes and texts that go in `appear as such – in a different font, on gray background`
- Results that come out in `appear as such — in the same font, on green background`
- Specific sections are `highlighted yellow as such` for emphasis
- The slides are designed for self-study as much as for the workshop
 - *accessible*, in substance and form, to go through on your own

The Workshop — Aims

- To make you aware what is possible with Git and GitHub
 - we will cover a large breath of issues, not all of it is for long-term memory
 - one reason why the slides are designed for self study as well
 - awareness of what is possible, Google, and perseverance are all we need
- To encourage you to convert into the Git and GitHub workflow
 - practice with a mock project, co-author
 - start converting a real one right after the workshop
 - happy to help via emails afterwards

Part 2. Tools

[Back to the contents slide.](#)

Workshop Materials — Overview

- Materials mimic a simple academic project
 - imagined to introduce a new dataset
 - on the Google Scholar rankings of fictitious journals
 - with a short manuscript
 - in the structure shown on the right

```
git_workshop-materials
|
|- data
|   |
|   |- journals.csv
|
|- manuscript
|   |
|   |- journals.docx
|   |- journals.pdf
|   |- journals.Rmd
|
|- README.md
```

Workshop Materials — Download from the Internet

- Download the materials from https://github.com/resulmit/git_workshop/tree/materials
 - on the webpage, follow
 - Code -> Download ZIP
- Unzip and rename the folder
 - unzip to a location that is not synced
 - e.g., perhaps Documents, but not Dropbox
 - rename the folder as YOURNAME_git_workshop
 - e.g., resul_git_workshop
 - this renaming will come handy later on

Workshop Materials — Contents — Data

`data\journals.csv`

- a dataset created with the `fabricatr` package (Blair et al., 2019), imagined to explore the Google Scholar rankings of fictitious journals
- includes the following variables
 - **name**: journals (1090 random titles)
 - **origin**: geographic origins (five continents)
 - **branch**: major discipline of journals (four branches)
 - **since**: time of first publication (years)
 - **h5_index**: H5 Index (integers)
 - **h5_median**: H5 Median (integers)
 - **english**: English (1) vs. other-language (0) journals
 - **subfield**: subfield (1) vs. generalist (0) journals
 - **issues**: number of issues published per year (integers)

Workshop Materials — Contents — Manuscript

- `manuscript\journals.docx`
 - the Microsoft Word document, holding the text for the manuscript of the project
- There are two other versions of `paper.docx` in the same folder
 - saved in PDF and R Markdown formats respectively
 - `manuscript\journals.pdf`
 - `manuscript\journals.Rmd`

Workshop Materials — Contents — README

README.md

- a Markdown document, holding the basic information about the project

Git — Download from the Internet and Install

- Git is a software that
 - keeps track of versions of a set of files
 - is *local* to you, the records are kept on your computer
 - is free
- To get this software
 - on Windows, install 'Git for Windows', downloading from <https://gitforwindows.org>
 - select 'Git from the command line and also from 3rd-party software'
 - on Mac, install 'Git', downloading from <https://git-scm.com/downloads>

GitHub — Open an Account

- GitHub is a hosting service, or a website, that
 - keeps a copy of the versions created by Git
 - is *remote* to you, like the Dropbox website
 - is, mostly, free
- To sign up for this service
 - register an account at <https://github.com>
 - registering an account is free
 - usernames are public, editable
 - either choose an anonymous username for now
 - or choose one carefully — it becomes a part of users' online presence

GitHub — Create a new GitHub repository

- Repository is a folder, or directory, which can keep files and other folders containing files under version control
 - a set of files whose records are kept together
 - repo is a popular abbreviation for repository

- To create a repo, on GitHub, follow:

```
Repositories -> New -> Repository name (e.g., "git_workshop") -> Public ->  
Create repository
```

- observe that
 - repository URLs have the following structure: https://github.com/USER_NAME/REPOSITORY_NAME
 - this is the address to view the repository online
 - for use in the Terminal, the address gets the .git extension
 - e.g., https://github.com/USER_NAME/REPOSITORY_NAME.git

GitHub — Create a new GitHub repository — Notes

There are two types of repositories

- public repositories
 - free for all to create
 - free for everyone else to see, copy, *propose* changes
- private repositories
 - in general, a paid service
 - with **GitHub Education**, it free for academics, students
 - requires a separate application

GitHub — Create a personal access token

- Accessing GitHub through the shell requires using a password
 - not the same password to login to GitHub website
 - personal access token (PAT) is the new generation of passwords
- Generate a PAT at <https://github.com/settings/tokens>
 - note it down for now
 - on how to caching PATs safely, see <https://happygitwithr.com/credential-caching.html>

GitHub Desktop — Download from the Internet and Install

- GitHub Desktop is a software that
 - makes working with Git and GitHub easier
- To get this software
 - download the file for your operating system from <https://desktop.github.com/>
 - and install

Other Resources*

- Git Reference Manual
 - available at <https://git-scm.com/docs>
- Pro Git (Chacon and Straub, 2021)
 - open access at <https://git-scm.com/book/en/v2>
- Git Cheat Sheet
 - available at <https://training.github.com/downloads/github-git-cheat-sheet.pdf>
- Git Visual Cheat Sheet
 - available at <https://ndpsoftware.com/git-cheatsheet.html>

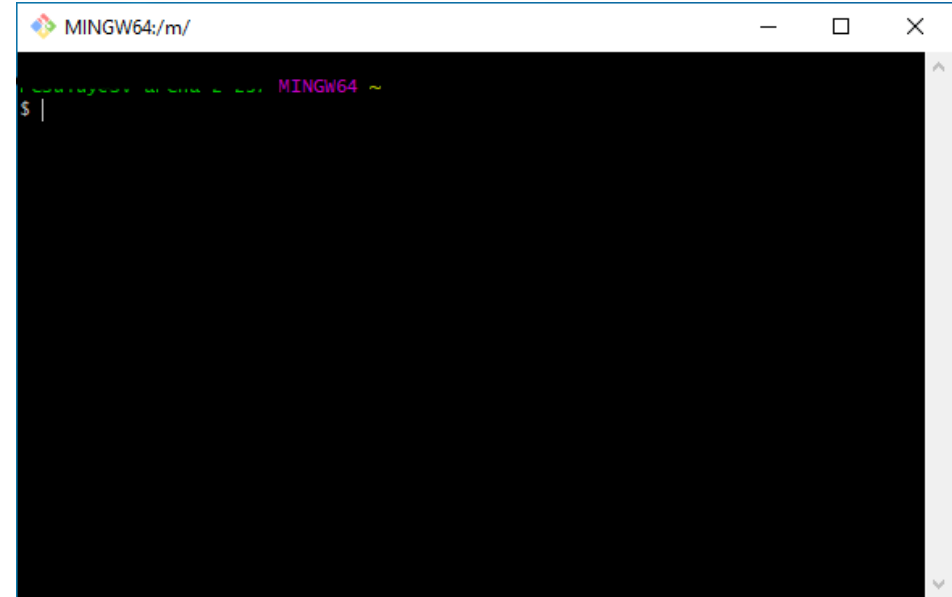
Part 3. Version Control

[Back to the contents slide.](#)

Version Control — Shell

Type, and enter, your commands into the shell

- also called command line interpreters
- depends on your operating system
 - on Windows, use the Git Bash app
 - on Mac, use the Terminal
- does not accept keyboard shortcuts
 - e.g., `ctrl + v` will not paste into shell
 - but mouse buttons and clicks will work



Version Control — Git Command Structure

Git commands

- start with the pre-fix `git`

```
$ git
```

Version Control — Git Command Structure

Git commands

- start with the pre-fix `git`
- might continue with
 - `a command`

```
$ git push
```

Version Control — Git Command Structure

Git commands

- start with the pre-fix `git`
- might continue with
 - a command
 - an option

```
$ git --version
```

Version Control — Git Command Structure

Git commands

- start with the pre-fix `git`
- might continue with
 - a command
 - an option
- might take
 - one or more arguments

```
$ git push origin master
```

Version Control — Git Command Structure

Git commands

- start with the pre-fix `git`
- might continue with
 - a command
 - an option
- might take
 - one or more arguments
 - one or more flags

```
$ git commit -m "I revised the abstract"
```

Version Control — config

- Use the `config` command to introduce yourself to Git
 - with the email address that you have used to sign up for GitHub

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

- enter the following line in the shell, to observe whether the previous step was successful

```
git config --global --list
```


Version Control — cd

Use the cd command to

- see which directory are you currently in

```
$ cd
```

```
/m
```

Version Control — cd

Use the cd command to

- see which directory are you currently in
- move into a directory that you prefer

Note that

- cd is not a git command
 - it does not start with git
- you can drag and drop the directory onto the shell
 - instead of typing the full path

```
$ cd ../pc/Desktop/resul_git_workshop
```

```
../pc/Desktop/resul_git_workshop
```

Version Control — `ls`

Use the `ls` command to list the contents of

- the current directory

```
$ ls
```

```
data/ manuscript/ README.md
```

Version Control — `ls`

Use the `ls` command to list the contents of

- the current directory
- a sub-directory

```
$ ls data/
```

```
journals.csv
```

Exercises

02:00

- 1) Move into the directory holding the workshop materials
 - using the `cd` command
 - hint: after a space, drag and drop the folder onto the shell

- 2) See the contents of the folder in the shell
 - using the `ls` command

Version Control — `git init`

Use the `git init` command to turn a directory into a git repository directory

```
$ git init
```

Initialized empty Git repository in `../resul_git_workshop/.git/`

Version Control — `git init` — Notes

The `git init` command creates a **hidden** `.git` subdirectory, with the structure on the right

- currently mostly empty
 - as nothing is being tracked yet
- not visible with `ls`
 - but visible with `ls -al`
 - on, on Windows, follow
 - File Explorer -> View -> Hidden items
- hidden, so that it is harder to delete or alter by mistake

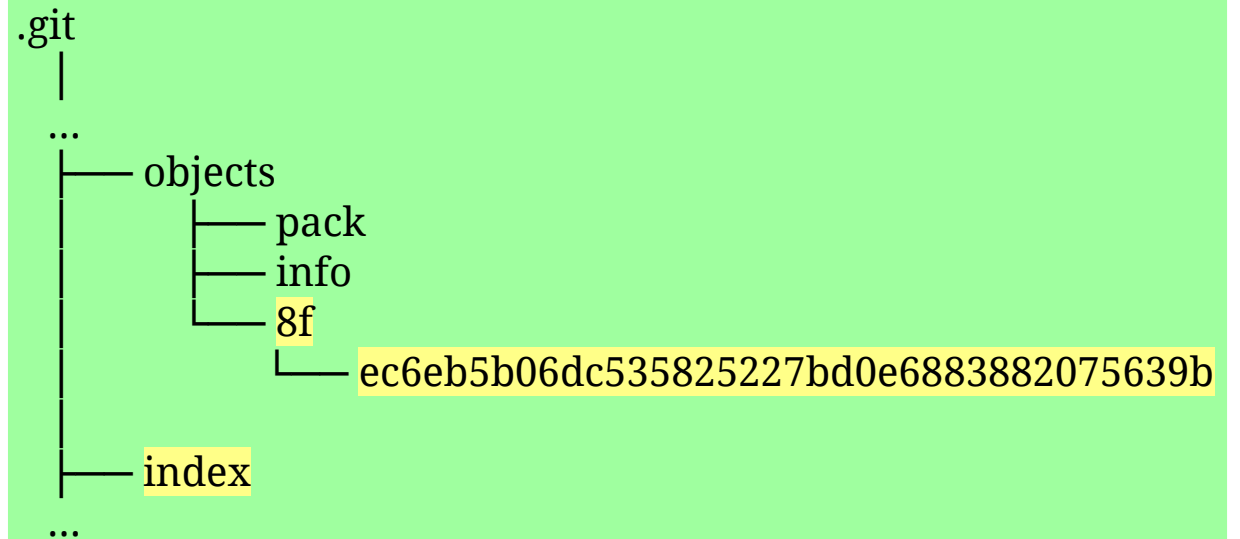
```
.git
├── refs
│   ├── tags
│   └── heads
├── objects
│   ├── pack
│   └── info
├── info
│   └── exclude
├── hooks
│   (12 files)
├── HEAD
├── config
└── description
```

Version Control — g i t add

Use the `git add` command to start tracking

- a specific file

```
$ git add README.md
```



Version Control — `git add`

Use the `git add` command to start tracking

- a specific file
- multiple files

```
$ git add README.md data/journals.csv
```

Version Control — g i t add

Use the g i t add command to start tracking

- a specific file
- multiple files
- everything in the directory

```
$ git add .
```

Version Control — `git add` — Notes

- The `git add` command alone does not create a version
 - it must be followed by the `git commit` command, without which
 - there is no version saved yet
 - the tracked files are said to be `.yellow-h[staged]`, or in the `.yellow-h[staging area]`
- Use the `git add` command (and then, the `git commit` command)
 - not only to add a new file to be versioned
 - but also, once that file changes, to snapshot a new version of that file every time
- A file in the staging area can be taken back, before its commit
 - by one of the following, depending on whether they have been versioned before

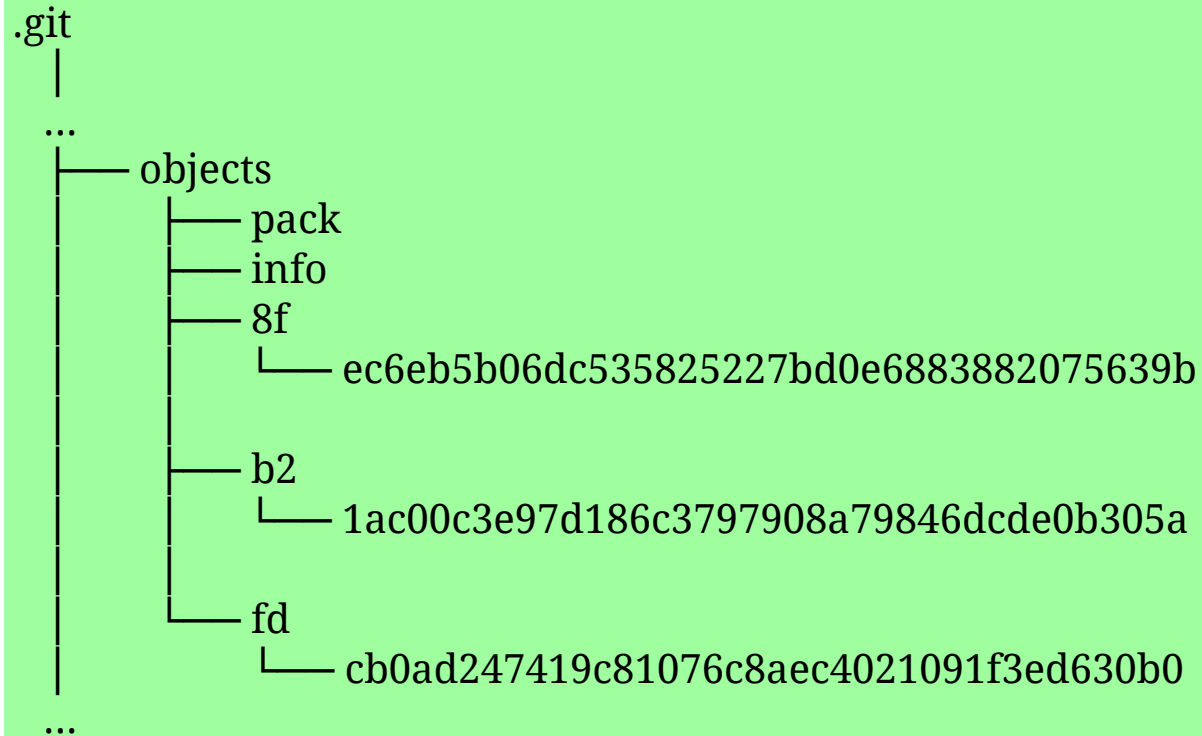
```
$ git rm --cached README.md
```

```
$ git rm --staged README.md
```

Version Control — `git commit`

Use the `git commit` command to take a snapshot of, or to version, a repository

```
$ git commit -m "adding the README file"
```



Version Control — `git commit`

Use the `git commit` command to take a snapshot of, or to version, a repository

- with the changes in the staging area

```
$ git commit -m "adding the README file"
```

Note that

- committing requires **registering a message**
 - to yourself and to co-authors to indicate what has changed

Version Control — `git commit`

Use the `git commit` command to take a snapshot of, or to version, a repository

- with the changes in the staging area

```
$ git commit -m "adding the README file"
```

Note that

- committing requires registering a message
 - to yourself and to co-authors to indicate what has changed
 - typed after the `-m` flag

Recall that

- committing requires one or more files in the staging area in the first place
 - staged with the `git add` command

Exercises

03:00

3) Start tracking the README.md file

- using the `git add` command

4) Take a snapshot of the current version of the file

- using the `git commit` command

Version Control — `git status`

Use the `git status` command to check the status of your repository

- e.g., are there any untracked, modified, deleted, staged etc. files?

```
$ git status
```

On branch master

Untracked files:

(use "git add ..." to include in what will be committed)

data/

manuscript/

no changes added to commit (use "git add" and/or "git commit -a")

Exercises

07:30

5) Check the status of your repository

- using the `git status` command

6) Make a change to the `README.md` file, and save

- e.g., edit the first sentence on the list

7) Check the status of your repository again

8) Commit the new version

- recall the combination of commands needed

Version Control — git log

Use the `git log` command to see the changes in reverse order

```
$ git log
```

```
commit 241b7285a00d47d69655d4a4afcd50989d5b50a8 (HEAD -> master)
```

```
Author: Resul Umit <resulumit@gmail.com>
```

```
Date: Sat Mar 6 09:27:32 2021 +0100
```

```
    correct the maths error
```

```
commit fdcb0ad247419c81076c8aec4021091f3ed630b0
```

```
Author: Resul Umit <resulumit@gmail.com>
```

```
Date: Sat Mar 6 08:55:21 2021 +0100
```

```
    adding the README file
```

Version Control — `git log`

Use the `git log` command to see the changes in reverse order

- the `oneline` flag returns a summary

```
$ git log --oneline
```

```
241b728 (HEAD -> master) correct the maths error  
fdcb0ad adding the README file
```

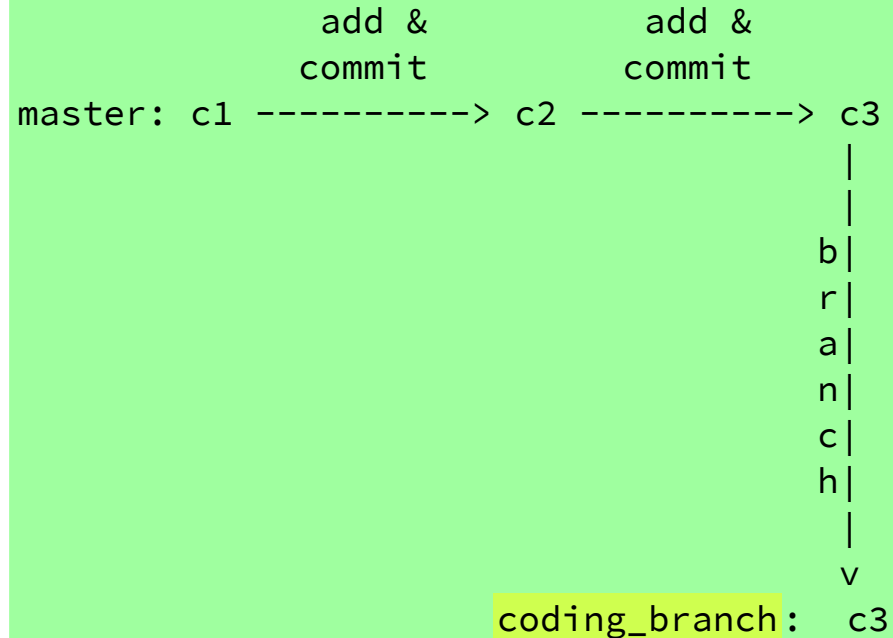
Note that

- commit hash is abbreviated to the first 7 digits, enough to uniquely identify each commit
 - i.e., 241b728 from 241b7285a00d47d69655d4a4afcd50989d5b50a8
- HEAD refers to the snapshot of your last commit
- master is the default branch name
 - we have only one branch at the moment, but can have multiple
 - think of a branch as a version of versions, or a copy of the whole repository

Version Control — g i t branch

Use the `git branch` command to create a copy of your repository, on your local machine

```
$ git branch coding_branch
```



Version Control — g i t branch

Use the `git branch` command to create a copy of your repository, on your local machine

```
$ git branch coding_branch
```

Note that

- this line takes an argument for **branch name**
 - the default repository is called master

Version Control — g i t branch

Use the `git branch` command to create a copy of your repository, on your local machine

```
$ git branch coding_branch
```

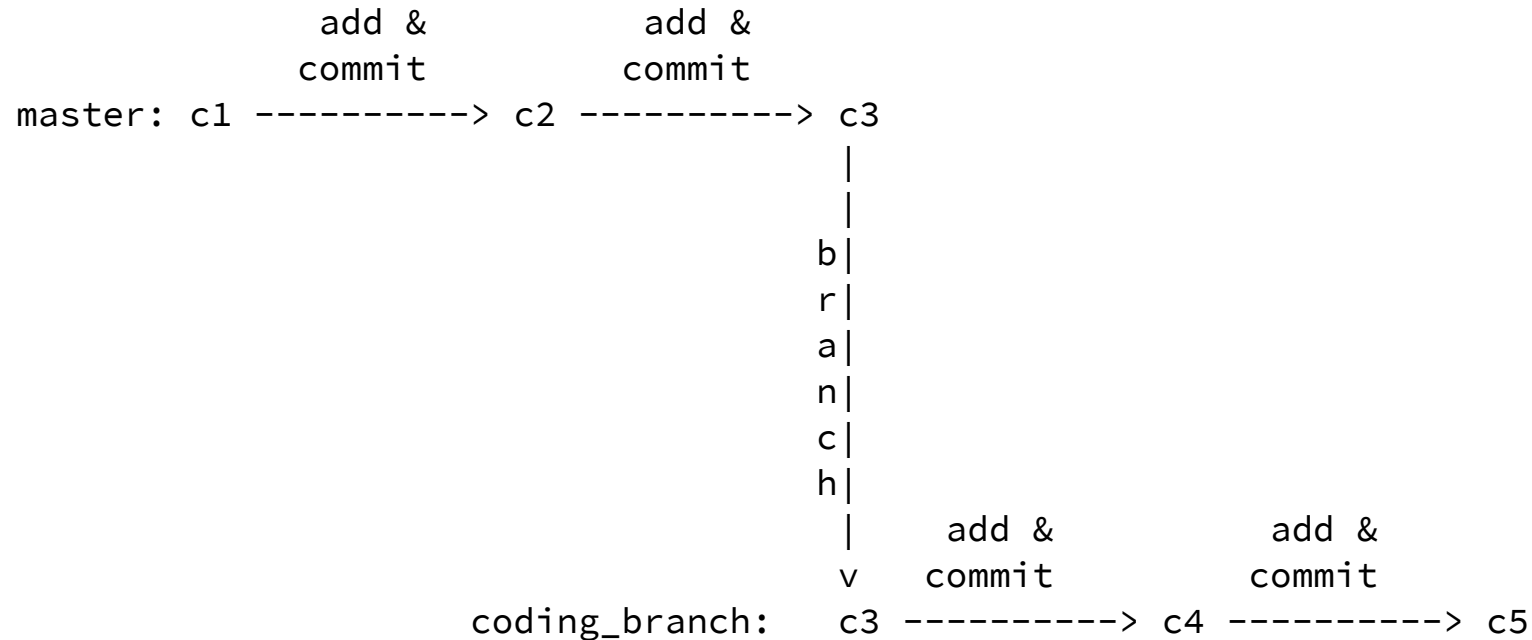
Note that

- this line takes an argument for branch name
 - the default repository is called master
- we are still in the master branch
 - check which branch are you in with the following command

```
$ git branch -v
```

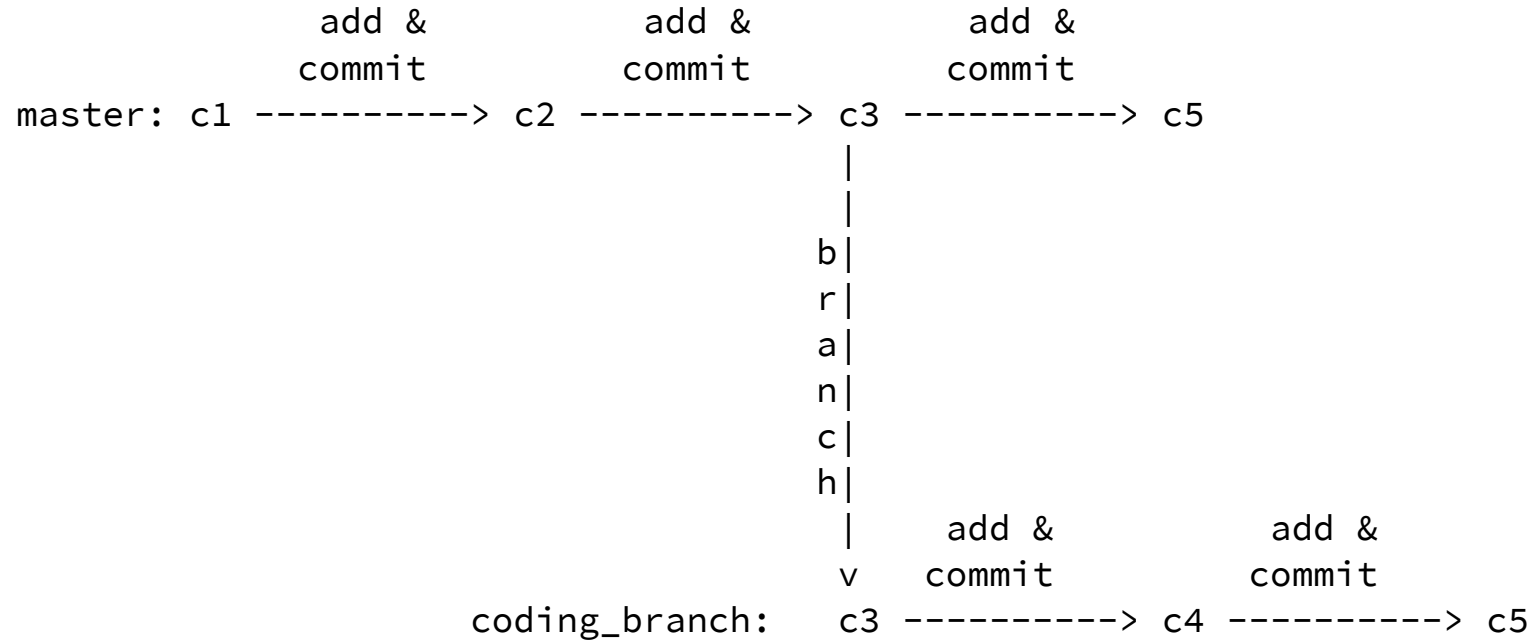
Version Control — g i t branch — Notes

- Branches can be developed
 - one at a time



Version Control — g i t b r a n c h — Notes

- Branches can be developed
 - one at a time
 - at the same time



Version Control — `git diff`

Use the `git diff` command to see what has changed, for changes

- not staged or committed yet

```
$ git diff
```

```
diff --git a/README.md b/README.md
```

```
...
```

```
--- a/README.md
```

```
+++ b/README.md
```

```
...
```

```
-1. Two times two makes five.
```

```
+1. Two times two makes four.
```

Version Control — `git diff`

Use the `git diff` command to see what has changed, for changes

- not staged or committed yet
- staged but not committed yet

```
$ git diff --staged
```

Version Control — `git diff`

Use the `git diff` command to see what has changed, for changes

- not staged or committed yet
- staged but not committed yet
- committed since a specific commit

```
$ git diff fdcb0ad
```

```
$ git diff HEAD~2
```

Version Control — `git diff`

Use the `git diff` command to see what has , for changes

- not staged or committed yet
- staged but not committed yet
- committed since a specific commit
- `between different commits`

```
$ git diff 241b728 1dfb2a5
```

```
$ git diff HEAD~1 HEAD~2
```

Version Control — `git diff`

Use the `git diff` command to see what has changed, for changes

- not staged or committed yet
- staged but not committed yet
- committed since a specific commit
- between different commits
- between different commits in one or more specific files

```
$ git diff 241b728 1dfb2a5 README.md
```

```
$ git diff HEAD~1 HEAD~2 README.md
```

Version Control — `git diff`

Use the `git diff` command to see what has changed, for changes

- not staged or committed yet
- staged but not committed yet
- committed since a specific commit
- between different commits
- between different commits in one or more specific files
- between different `branches`

```
$ git diff master..coding_branch
```

Version Control — g i t checkout

Use the g i t checkout command to switch to

- a different version of all files in the same branch

```
$ git checkout 241b728
```

```
$ git diff HEAD~1
```

Version Control — git checkout

Use the git checkout command to switch to

- a different version of all files in the same branch
- a different version of one or more specific files

```
$ git checkout 241b728 README.md
```

```
$ git checkout HEAD~1 README.md
```


Version Control — g i t checkout

Use the g i t checkout command to switch to

- a different version off all files in the same branch
- a different version of one ore more specific files
- a different branch

```
$ git checkout coding_branch
```

Exercises

07:30

9) Make sure there is nothing to commit in the master branch

- using the `git status` command to see

10) Make a new branch and switch to it

- using the `git branch` and `git checkout` commands

11) Start tracking the `data/journals.csv` file in this new branch

12) Commit the new version to this branch

13) Rename the first variable of the dataset, stage, and commit

- e.g., from `name` to `journal_name`
- both in `README.md` and `data/journals.csv`

14) Switch back to the master branch

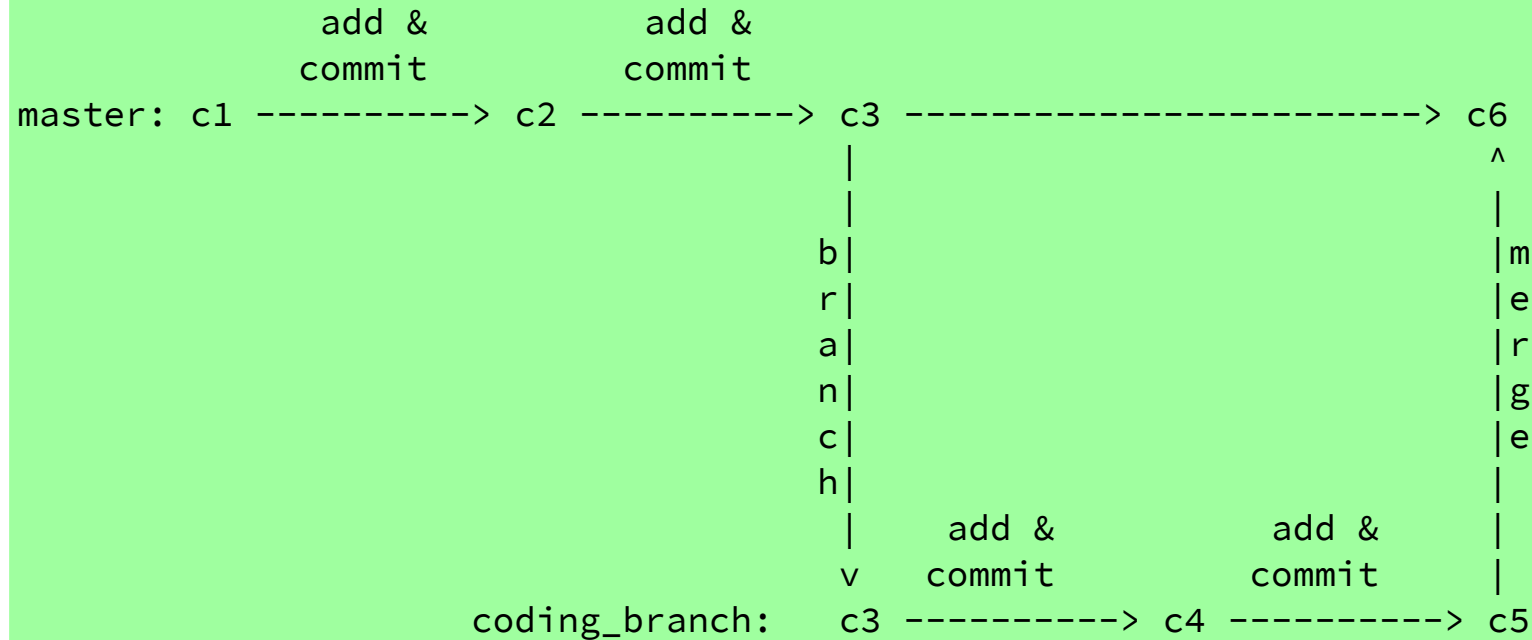
15) Check the differences between the branches

- using the `git diff` command

Version Control — `git merge`

Use the `git merge` command to integrate different versions of your repository into a single version

```
$ git merge coding_branch
```



Version Control — g i t merge

Use the `git merge` command to integrate different versions of your repository into a single version

```
$ git merge coding_branch
```

Note that

- this line includes one argument
 - `from where` to merge

Version Control — `git merge`

Use the `git merge` command to integrate different versions of your repository into a single version

```
$ git merge coding_branch
```

Note that

- this line includes one argument
 - `from where` to merge
- this merges into the current branch
 - move into the desired branch first
 - using the `git checkout` command

Version Control — `git merge` — Conflicts

- Merge is done automatically if there are no conflicts
 - conflicts emerge when versions to be merged include edits *on the same line of the same file*
 - edits on different lines are not a problem as changes are tracked line by line
- Merge conflicts require human intervention
 - by editing the conflicting lines
 - after the merge attempt
 - marked in the document with the conflict
 - and committing the corrected file

```
<<<<<<< HEAD
1. Two times two makes five.
=====
1. Two times two makes tree.
>>>>>>> new_branch
```

Exercises

07:30

- 16) Merge the coding_branch into the master branch
- 17) Make a change on the same line of the README.md in both branches
- 18) Try to merge the coding_branch into the master branch
 - hint: this will lead to a merge conflict
- 19) Solve the conflict and commit

Version Control — `git remote add`

Use the `git remote add` command to define a different place, to save an backup of your repositories

- e.g., a repository on GitHub
 - could also be a directory on an external hard drive

```
$ git remote add origin https://github.com/resulumit/git_workshop.git
```


Version Control — `git remote add`

Use the `git remote add` command to define a different place, to save an external copy of your repositories

- e.g., a repository on GitHub
 - could also be a directory on an external hard drive

```
$ git remote add origin https://github.com/resulumit/git_workshop.git
```

Note that this includes

- a valid address for the external copy
 - e.g., there is a repository on my GitHub account called `git_workshop`

Version Control — `git remote add`

Use the `git remote add` command to define a different place, to save an external copy of your repositories

- e.g., a repository on GitHub
 - could also be a directory on an external hard drive

```
$ git remote add origin https://github.com/resulumit/git_workshop.git
```

Note that this line includes

- a valid address for the external copy to be saved
 - e.g., there is a repository on my GitHub account called `git_workshop`
- an arbitrary name for this external repository
 - traditionally called `origin`, could also be anything else

Version Control — `git push`

Use the `git push` command to save a copy of your `local` repository to its `remote` repository

```
$ git push origin master
```

Version Control — git push

Use the git push command to save a copy of your local repository to its remote repository

```
$ git push origin master
```

Note that

- this line includes two arguments
 - where to push
 - already defined with the git remote add command

Version Control — git push

Use the git push command to save a copy of your local repository to its remote repository

```
$ git push origin master
```

Note that

- this line includes two arguments
 - where and what to push
 - could be any branch

Version Control — git push

Use the git push command to save a copy of your local repository to its remote repository

```
$ git push origin master
```

Note that

- this line includes two arguments
 - where and what to push
- pushing will be rejected if the remote repository has edits that the local repository does not
 - necessitating a pull first

Version Control — `git push`

Use the `git push` command to save a copy of your local repository to its remote repository

```
$ git push origin master
```

Note that

- this line includes two arguments
 - where and what to push
- pushing will be rejected if the remote repository has edits that the local repository does not
 - necessitating a pull first
- if this is your first time using GitHub, you will be prompted to authenticate
 - follow the instructions on your screen and in your email
 - using the PAT that you created in Part 2

Version Control — `git pull`

Use the `git pull` command to get a copy of the `remote` repository, and merge it into the `local` repository

```
$ git pull origin master
```


Version Control — git pull

Use the `git pull` command to get a copy of the remote repository, and merge it into the local repository

```
$ git pull origin master
```

Note that

- this line includes two arguments
 - `where` to pull from
 - already defined with the `git remote add` command

Version Control — `git pull`

Use the `git pull` command to get a copy of the remote repository, and merge it into the local repository

```
$ git pull origin master
```

Note that

- this line includes two arguments
 - where and **what** to pull from
 - could be any branch

Version Control — `git pull`

Use the `git pull` command to get a copy of the remote repository, and merge it into the local repository

```
$ git pull origin master
```

Note that

- this line includes two arguments
 - where and what to pull from
- pulling involves fetching and merging
 - might lead to conflicts, and necessitate solving them manually

Exercises

07:30

20) Add a remote repository

- e.g., the **GitHub repository that you have already created**
- using the `git remote add` command

21) Push the master branch to GitHub

- using the `git push` command
- observe the repository on GitHub

22) Commit a change to the README.md file on GitHub

- by clicking on the pen symbol

23) Pull the changes to the local repository

- using the `git pull` command

Part 4. What to Version

[Back to the contents slide.](#)

What to Version — Overview

There might be good reasons to exclude some others from versioning

- for local-only repositories
 - files that we (re-)create automatically as outputs
 - e.g., `paper.pdf`, as opposed to `paper.Rmd`
- for repositories that are also on GitHub
 - files that contain information that we do not want others to see
 - e.g., personal API keys
 - files that we do not have the right to share with others
 - e.g., secondary data with user agreements otherwise
 - files that are too large
 - individual files cannot be larger than 100MB

What to Version — .gitignore

- .gitignore specifies which file(s) and/or folder(s) should be excluded from version control
 - is a file itself, to be saved in the root directory
 - allowing for us to use the `git add .` shortcut
- .gitignore lists one item per line
 - each line has a pattern, which determines whether one or more files or folders are to be ignored
- See these
 - documentation at <https://git-scm.com/docs/gitignore>
 - templates at <https://github.com/github/gitignore>

What to Version — .gitignore — Examples

You can ignore, for example,

- a specific folder, relative to the root directory

```
/manuscript/
```


What to Version — .gitignore

You can ignore, for example,

- a specific folder, relative to the root directory
- a specific file in a specific folder, relative to the root directory

```
/manuscript/  
/manuscript/paper.pdf
```

What to Version — .gitignore

You can ignore, for example,

- a specific folder, relative to the root directory
- a specific file in a specific folder, relative to the root directory
- a specific file in any folder

```
/manuscript/  
/manuscript/paper.pdf  
paper.pdf
```

What to Version — .gitignore

.pull-left[

You can ignore, for example,

- a specific subdirectory, relative to the root directory
- a specific file in a specific subdirectory, relative to the root directory
- a specific file in any subdirectory
- all files with a specific extension, anywhere in the repository

```
/manuscript/  
/manuscript/journals.pdf  
journals.pdf  
*.pdf
```

What to Version — .gitignore — Notes

- There are many other pattern formats
 - see the documentation at <https://git-scm.com/docs/gitignore>
- Starting to ignore a file or folder that is already being tracked requires clearing the cache
 - after changing and saving .gitignore, enter the following line in the shell
 - with your specific /path/to/file

```
git rm --cached /path/to/file
```

- The following command clears *all* cache
 - might be useful after changes to .gitignore that involves several files or folders
 - but should be used with care, on an otherwise up-to-date repository

```
git rm -r --cached .
```

Exercises

07:30

24) Create a `.gitignore` file on GitHub

- to ignore the `paper.pdf` file
- save and commit

25) Pull the changes to the local repository

- using the `git pull` command

26) Start tracking everything

- using the `git add .`
- and commit

27) Push the changes to the remote repository

- observe the repository on GitHub to confirm `.gitignore` work as intended

Part 5. Collaboration

[Back to the contents slide.](#)

Collaboration — Project Setup

- The setup depends on the users' role, on whether they are
 - the *owner* who creates the GitHub repository, or
 - the *collaborator* who is then added to that repository
- Once the project is setup
 - it continues to be associated with the owner's GitHub profile
 - at the same time, it is listed under the collaborator's profile as well
 - both the owner and the collaborator have the same rights, unless otherwise restricted

Collaboration — Project Setup — Owner

- The setup for the owner is largely the same as in any single-author, single-computer scenario
 - introduce version control to a local project with Git,
 - create a remote repository for that project on GitHub, and
 - associate the local project with the remote repository
- As an additional step, the owner needs to invite their collaborator(s) to the project
- following, from the relevant GitHub repository,
 - ▮ Settings -> Manage access -> Invite a collaborator

Collaboration — Project Setup — Collaborator

- Notice that the remote part of the setup is done by the owner for the collaborator
 - subject to acceptance of the invitation
 - invitations are available directly at <https://github.com/notifications>, but also sent via email
 - on acceptance, projects appear among the repositories of the collaborator
- The local part of the setup still needs to be done
 - by using the `git clone` command with the repository address

```
$ git clone https://github.com/USER_NAME/REPOSITORY_NAME.git
```

Exercises

10:00

28) Prepare for collaboration

- by assuming the role of first and second co-authors in your group
- Owners:
 - invite your activity party ion to collaborate
 - hint: you will need their username, full name, or email address to do so
- Collaborators:
 - accept the invitation from your co-author
 - clone the project to your a local repository
 - using the `git clone` command

Colloboration — Workflow

1) Pull

- on the Git tab in RStudio, click Pull to move the up-to-date records from GitHub to your computer
 - if your collaborator has not pushed anything since your last pull, you will be noticed that Already up-to-date.
 - collaborative projects require pulling as well as pushing because your collaborator(s) might have pushed their commits to GitHub
 - pulling frequently minimises the risk of merge conflicts

2) Edit and save; commit and push

- the same procedure as in any single-author, single-computer scenario
 - as described on [this slide](#) forward
- pushing frequently minimises the risk of merge conflicts

Exercise

05:00

29) Non-simultaneous Collaboration

- take in turns with your partner to work on the same document (of the same project)
 - *owner*: edit the first header in the document (i.e., "R Markdown"), save, commit, and push
 - *owner and collaborator*: observe the changes, if any, on your own .Rmd file, and/or on your GitHub repository
 - click on the relevant commit message on GitHub and observe the commit
 - *collaborator*: pull, revert the header back to original, save, commit, and push
- notice that you have not encountered any errors and/or merge conflicts
 - because everyone edited and merged with an up-to-date document
 - this is the default scenario in single-author, multiple computer scenario

Exercise

10:00

30) Simultaneous Collaboration — Different Lines

- work on the same document at the same time
 - *owners*: edit the first header in a document, save, commit, and push
 - *collaborators*: edit the second header in the document (i.e., "Including Plots"), save, commit, and push
 - observe the error message that the last pusher will receive, follow the instructions on RStudio to solve the problem
- notice that you have encountered an error but not a merge conflict
 - pulling before pushing solves the problem because the edits are not on the same line
 - the merge takes place automatically, on the *local* repository of the last pusher

Exercise

10:00

31) Simultaneous Collaboration — Same Line

- work on the same document at the same time
 - *owners*: edit the first header in the document again, save, commit, and push
 - *collaborators*: edit the first header in the document as well, save, commit, and push
 - observe the error message that the last pusher will receive, follow the instructions on RStudio to solve the problem
- notice that you have encountered not only an error but also a merge conflict
 - pulling before pushing alone does not solve the problem because the edits are on the same line
 - the conflict cannot be solved automatically — it needs human intervention
 - by pulling first, you can view the conflict directly on the file
 - marked between less than < and greater than > signs, divided by the equal signs
 - solution is to accept the remote version, by deleting your edit and or moving that edit to a different line
 - merging takes place on the *local* repository of the last pusher

Colloboration — Workflow — Alternative

- The workflow above is rather simple, but has some disadvantages, including
 - not easy, albeit still possible, to see the edits of the collaborators
 - not clear who is in charge of the overall progress
 - not possible to discuss edits
 - not possible to compromise on conflicting edits
- An alternative workflow exists
 - work on different branches of the same project
 - version control to your own branch
 - create pull requests with comments
 - merge the branch into master

Colloboration — Workflow — Alternative

1) Branch


- using the `git branch` command to create a branch

2) Edit and save; commit and push

- the same procedure as in any single-author, single-computer scenario
- except that now pushing into a branch
- notice, on GitHub, that your commit is in the new branch, while *master* remains unchanged

3) Pull request

- On GitHub, click

 Pull requests -> New pull request

- choose what is to be pulled, and write a note to your collaborator who can accept or reject the merge
 - if there are merge conflicts, the collaborator solves them on GitHub before merging

Exercises

10:00

32) Pull request

- create a pull request for your collaboration project
 - create a branch for yourself
 - edit any line, save, commit, and push
 - request your branch to be merged

33) Merging

- view the pull request of your collaborator on GitHub
- merge it to *master*

Part 6. Third-Party Applications

[Back to the contents slide.](#)

Applications — RStudio

Integrating RStudio into the workflow requires

1. **initial setup**, done once^{*}
 - unless for a new computer or, if ever, a new GitHub account
 - a bit technical, but worth the hassle
2. **project setup**, repeated for every project
 - shorter, less complicated

^{*} We have started this process already, in Part 1 of the workshop, by downloading and installing Git and signing up for GitHub. [Back to the relevant slide.](#)

Applications — RStudio — Initial Setup

1) Enable version control with RStudio

- from the RStudio menu, follow:

Tools -> Global Options -> Git/SVN -> Enable version control interface for RStudio projects

- RStudio will likely find Git automatically. In case it cannot, Git is likely to be at
 - `c:/Program Files/Git/bin/git.exe` on Windows
 - `/usr/local/git/bin/git` on Mac

Applications — RStudio — Initial Setup

2) Set Git Bash as your shell (Windows-only step)

- from the RStudio menu, follow:

Tools -> Global Options -> Terminal -> New terminals open with: Git Bash

Applications — RStudio — Initial Setup*

3) Introduce yourself to Git

- from the RStudio menu, follow:

Tools -> Terminal -> New Terminal

- enter the following lines in the Terminal, with the email address that you have used to sign up for GitHub

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

- enter the following line in the Terminal, to observe whether the previous step was successful

```
git config --global --list
```

* This repeats the process on [this slide](#) in Part 3. Repeating it is unnecessary, but will not cause any problems.

Applications — RStudio — Project Setup

1) Create an RStudio project

- RStudio allows for dividing your work with R into separate projects, each with own history etc.
 - [this page](#) has more information on why projects are recommended
- Create a new RStudio project, following from the RStudio menu:

File -> New Project

Applications — RStudio — Project Setup

2) Initiate local version control with Git

- from the RStudio menu, follow:

```
Tools -> Version Control -> Project Setup... -> Version Control System -> Git
```

- after confirming your new repository, and restarting the session, observe that
 - now there is now a Git tab in RStudio, with buttons for various git commands
 - your project now includes a `.gitignore` file
 - untracking some project-specific files

Applications — RStudio — Project Setup

3) Create a new GitHub repository

- on GitHub, follow:

```
Repositories -> New -> Repository name (e.g., "git_workshop") -> Public ->  
Create repository
```

- observe that
 - repository URLs have the following structure: https://github.com/USER_NAME/REPOSITORY_NAME
 - this is the address to view the repository online
 - for use in the Terminal, the address gets the .git extension
 - e.g., https://github.com/USER_NAME/REPOSITORY_NAME.git

Applications — RStudio — Project Setup

3) Push an existing repository for the first time

- from the RStudio menu, follow:

Tools -> Terminal -> New Terminal

- enter the following lines in the Terminal, with your username and repository name

```
git remote add origin https://github.com/USER_NAME/REPOSITORY_NAME.git
git add .
git commit -m "first commit"
git push -u origin master
```

- if this is your first time using GitHub with RStudio, you will be prompted to authenticate
 - follow the instructions on your screen and in your email
- observe that your project files are now online, listed on the GitHub repository

Applications — RStudio — Notes

- The principles covered in **Part 3** to **Part 5** remain the same
 - e.g., add, commit, and push
- There are now two alternative ways to proceed
 - typing git commands in the Terminal
 - as opposed to in a stand alone shell
 - clicking the buttons on the Git tab
 - e.g., Diff, Commit

References

[Back to the contents slide.](#)

References

Blair, G., Cooper, J., Coppock, A., Humphreys, M., Rudkin, A. and Fultz, N. (2019). **fabricatr: Imagine your data before you collect it**. R package, version 0.10.0.

The workshop ends here.

Congratulations for making it this far, and
thank you for joining me!

[Back to the contents slide.](#)