

Introduction to Workflow and Git(Hub)

Best practice: Folder and file structure

1. separate folders for scripts, data, output, and reports
2. raw data files separate from processed data
3. clear and consistent names for script and output files
4. numbering, lowercase, and separate words with underscores or hyphens
5. if date necessary, usually in the end, sort by YYYYMMDD
6. multiple script files for different (sub) tasks (max 100 lines)

Best practice: Efficient R scripts

1. define libraries, default variables, source code at top of script
2. comment and structure sections (# ---- headline ----)
3. use pipe operator |> (magritter: %>%) for combining functions
4. use indentations and spaces for readability
5. **DRY** - use lists, lapply, vectorization, and functions
6. use relative paths for data and output
7. avoid hard coded subsetting and indexing

Best practice: Resources

- Best Coding Practices for R
- Structuring R projects
- R Best Practices
- Tips for organising your R code
- Nice R Code
- Repeating things: looping and the apply family

Code Editor: VSCode

The screenshot shows the Visual Studio Code interface with several panes:

- Left Sidebar:** Shows the R workspace with objects like `dt`, `model`, and `n`. It also includes help pages and live share controls.
- Top Left Editor:** The file `data_analysis.R` contains R code for generating data, fitting a linear model, and plotting residuals. A portion of the code is as follows:

```
library(data.table)
set.seed(123)
n <- 1000
dt <- data.table(id = 1:n)
dt[, x1 := rnorm(.N, mean = 0, sd = 2)]
dt[, x2 := runif(.N, min = -1, max = 1)]
dt[, y := 2 * x1 + x2 + 0.5 * rnorm(.N)]
model <- lm(y ~ x1 + x2, data = dt)
summary(model)
plot(model)
```

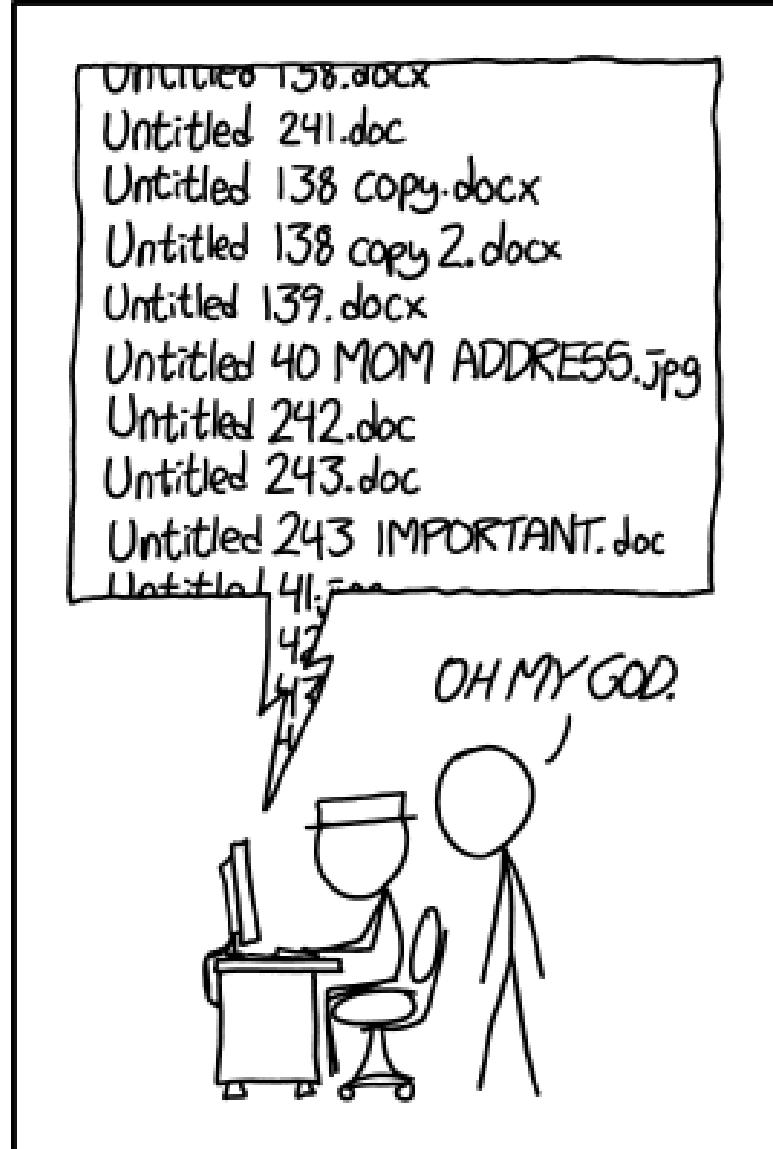
- Bottom Left Editor:** An `R Plot` pane displays a scatter plot titled "Residuals vs Leverage". The y-axis is "Standardized residuals" and the x-axis is "Leverage $\text{lm}(y \sim x1 + x2)$ ". The plot shows many data points with two outliers labeled `557` and `2850`. A horizontal red line represents the zero residual level, and a dashed red line indicates Cook's distance.
- Right Editor:** The terminal window shows the R session output for running the `data_analysis.R` script. It includes the R version information, library loading, data generation, model fitting, and summary statistics. The output ends with the command `> plot(model)`.
- Bottom Status Bar:** Displays the R version (4.1.3), line count (Ln 14, Col 1), and other status information.

Code Editor: Benefits

- Swiss army knife of coding and file management
 - search (and replace) in whole project folder
 - side-by-side editor windows
 - better file and folder management
 - customizable (with extensions)
- multiple languages supported (e.g. R, Python, LaTeX, Markdown)
- Git(Hub): easy integration for better workflow
- with R:
 - run multiple R Sessions in parallel
 - scripts still editable if process busy

Code Editor: Resources

- <https://code.visualstudio.com/docs/languages/r>
- <https://renkun.me/2019/12/11/writing-r-in-vscode-a-fresh-start/>
- <https://schiff.co.nz/blog/r-and-vscode/>
- <https://rolkra.github.io/R-VSCode/>



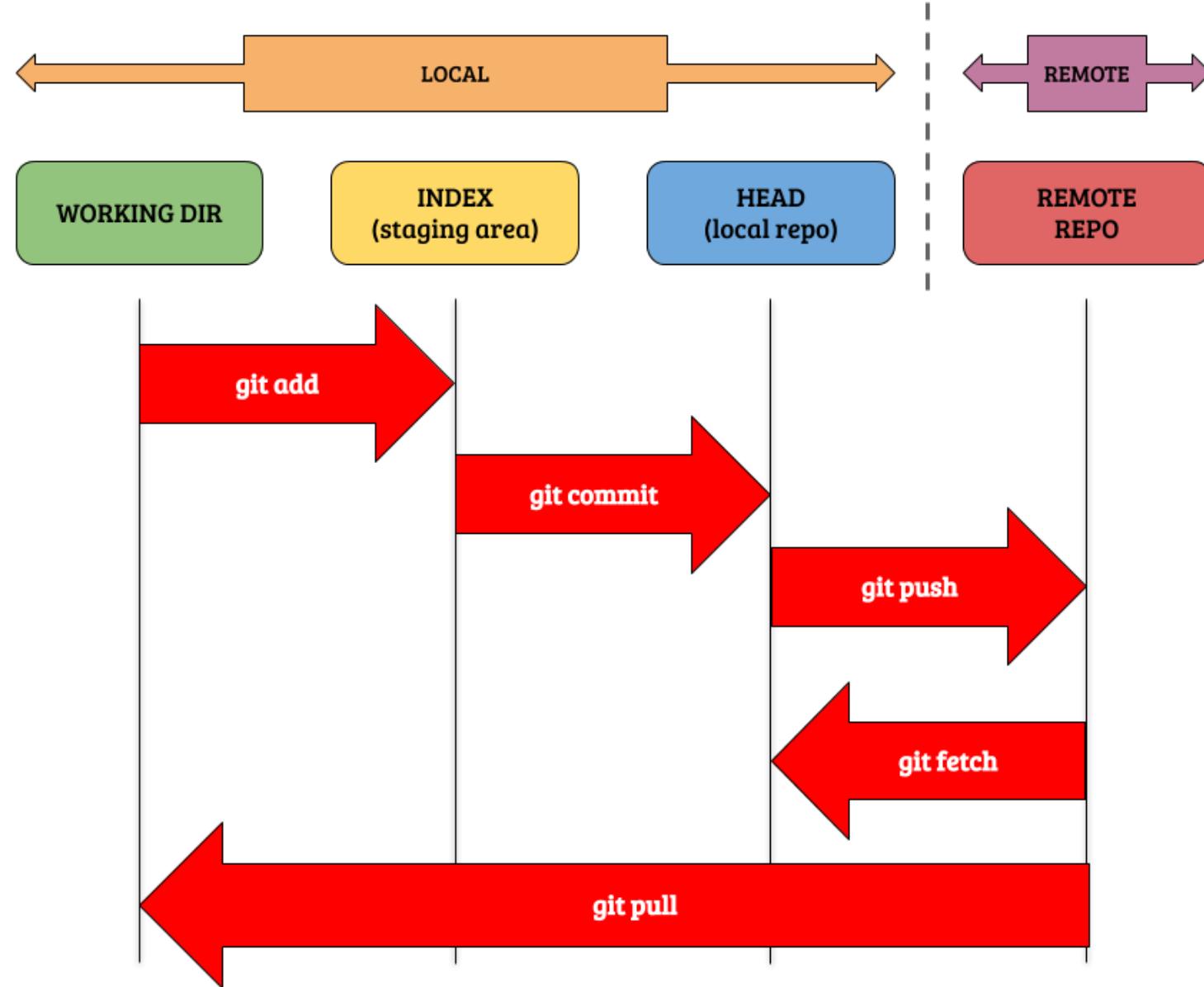
PROTIP: NEVER LOOK IN SOMEONE
ELSE'S DOCUMENTS FOLDER.

Git: Idea and concept

- distributed version control system
- track and document changes in code
- compare and find differences
- "time travel machine"
- helps to imagine changes as smaller tasks
- share, publish, and collaborate on projects

Git: How to use it

	Ease of use	Efficiency	Nerdiness
CLI	● ○ ○ ○ ○	● ● ○ ○ ○	● ● ● ● ●
GUI	● ● ● ○ ○	● ● ● ○ ○	● ● ○ ○ ○
RStudio	● ● ● ● ○	● ● ● ○ ○	● ● ● ○ ○
VSCode	● ● ● ● ●	● ● ● ● ○	● ● ● ● ○



Git: First and basic steps

```
$ git config --global user.name <your name>  
$ git init <your repository name>  
$ git status
```

```
$ git add <file-name-1> <file-name-2> OR --all  
$ git commit -m "<commit-message>"  
OR BOTH IN ONE  
$ git commit -am "<commit-message>"
```

GitHub: Remote and cooperative workflow

```
$ git clone <git-repo-url>
```

```
$ git branch < branch-name>
```

```
$ git checkout <name-of-your-branch>
```

OR BOTH IN ONE

```
$ git checkout -b <name-of-your-branch>
```

```
$ git push -u (<remote> OR origin) <branch-name>
```

```
& git fetch
```

```
& git merge <branch-name>
```

OR BOTH IN ONE

```
$ git pull <remote> (<branch-name>)
```

GitHub: Credentials

- for GitHub remote commands: username and personal access token needed
- Settings → Developer Setting → Personal Access Tokens → Generate new token
- enter username and PAT everytime, or:
 - i. `git config credential.helper store`
 - ii. use VSCode

GitHub: .gitignore

Why: text file with folders and files (patterns) not to track

What: sensitive data; temp and old files; big data files; outputs

→ usually track just plain text files (e.g. R scripts, TeX source, etc.)

How: 2 approaches ([helpful online tool creates content automatically](#))

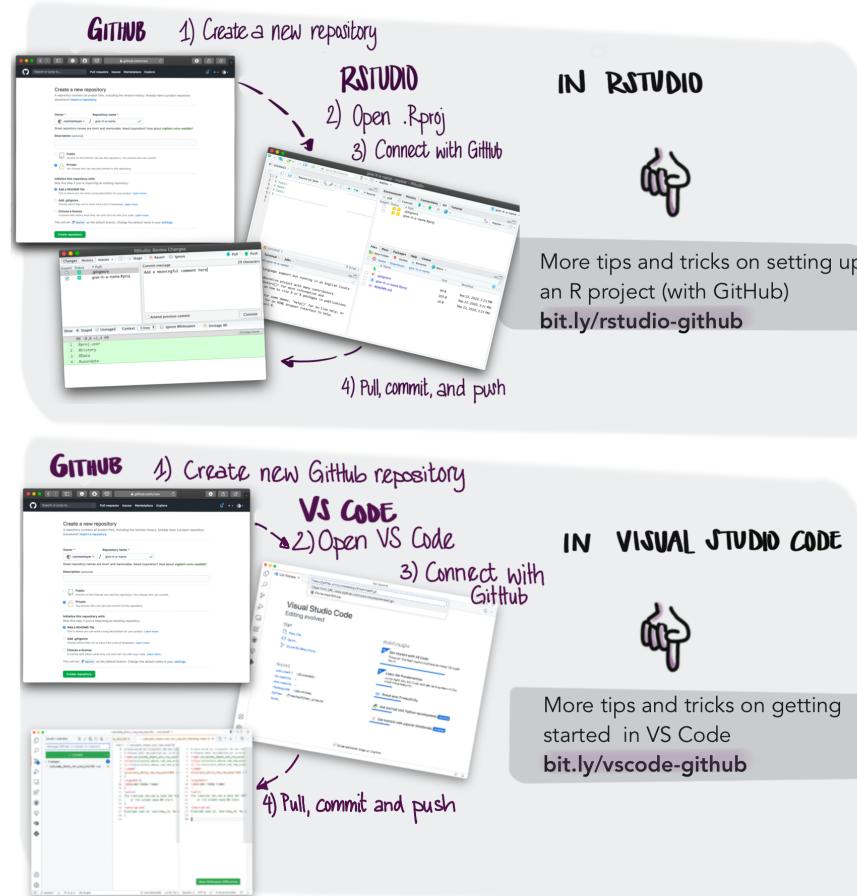
```
/data/old  
passwords.txt  
*.doc  
-----OR-----  
/*  
!.gitignore  
!/scripts
```

Git: Ressources

- [Intro to Git, for the Social Scientist](#)
- [Git for Social Scientists](#)
- [Git for Students in the Social Sciences](#)
- [GitHub - The Perks of Collaboration](#)
- [AI tool suggesting git command](#)

VERSION CONTROL + COLLABORATE

SET UP GITHUB



GitHub: Course material

1. install git
2. create a GitHub account
3. tell user name to become a collaborator
4. clone our course material **repository**
5. add a personal folder and test file
6. push this changes to the remote repository
7. pull changes of the other participants

Cosima-meyer