Intro to Programming with R for Political Scientists

Session 1: R-Studio and Version Control

Markus Freitag

Geschwister Scholl Institute of Political Science, LMU

2021-05-22

Intro

Welcome Young Padawans

Why this course?

- Intro to R with a focus on programming (<-> standard pol sci quant training).
- Set some good programming/workflow habits to make your life easier later on.

> *"I want to give a short (crash) course that provides you with some basic practical skills necessary to get serious about (political) sciencing."*

- There are tons of awesome & free materials from the best R Gurus of the world out there.
- Take this course as a humble starting point to see through the thickit.

Welcome Young Padawans

Who am I?

- Soon™ PhD student. Likes stats and pol sci. Also likes cooking and (board) gaming.
- Web: 🌐

Who are you?

I hate online introductory rounds.

Let's do a break-out session instead so you get to know a few people.

This will also be your team for the problem sets.

What are we going to cover?

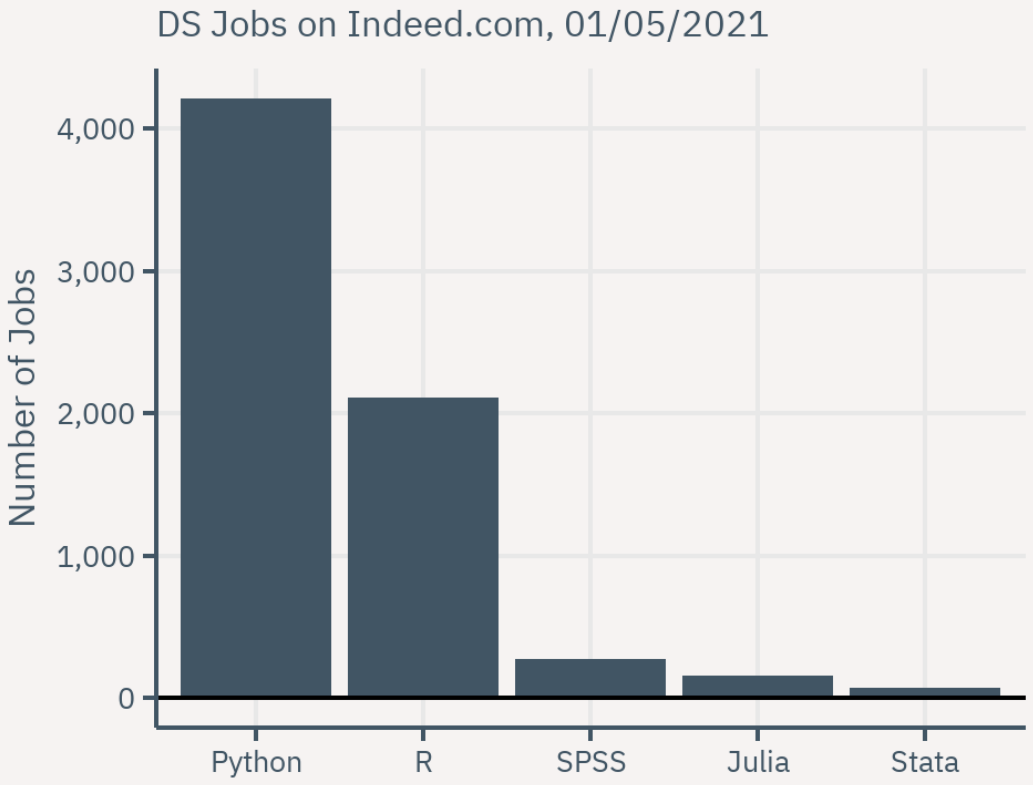
1. **Intro**
2. **R-Studio and (Git)Hub**
3. **Base R & Tidyverse Basics**
4. **Data Wrangling I**
5. **Data Wrangling II**
6. **Data Viz**
7. **Writing Functions**
8. **A complete scientific workflow with R**

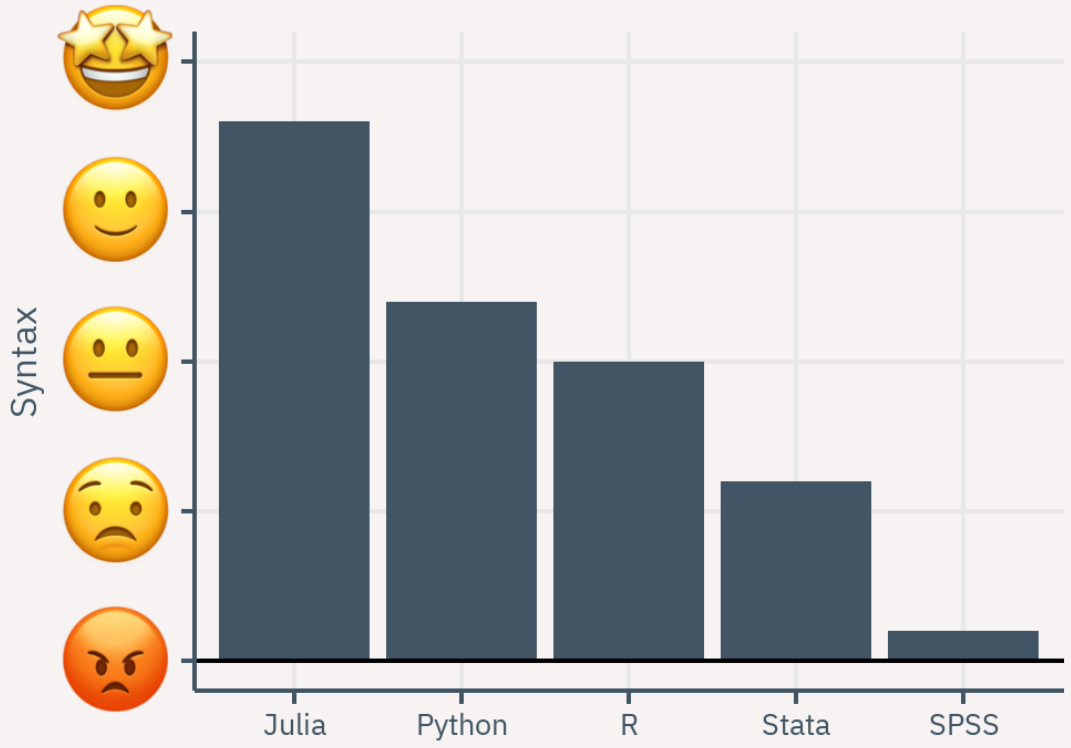
Why R?

At the present day (and surely years to come), R is arguably the best programming language for academics:

- R is from statisticians for statisticians.
- Most active (academic) development community for **statistical** computing/programming.
- Nice IDEs; Good integration of other languages and workflow components.
- Best for data viz.

Why R?


DS Jobs on Indeed.com, 01/05/2021
Number of Jobs
4,000
3,000
2,000
1,000
0
Python
R
SPSS
Julia
Stata


Syntax
Julia
Python
R
Stata
SPSS

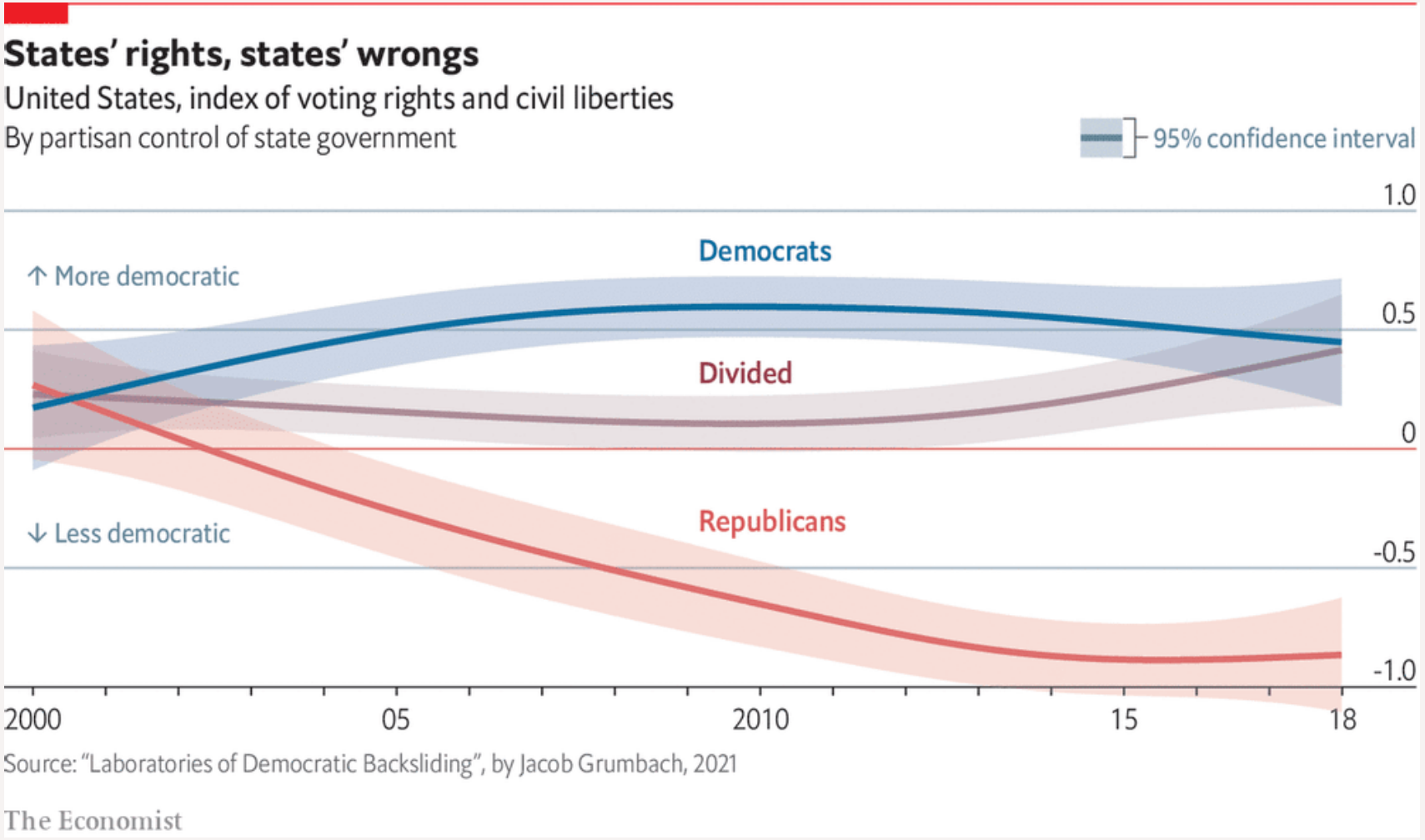
R versus other langs/software

Showcase

U can make pretty graphs...

Showcase

U can make pretty graphs...

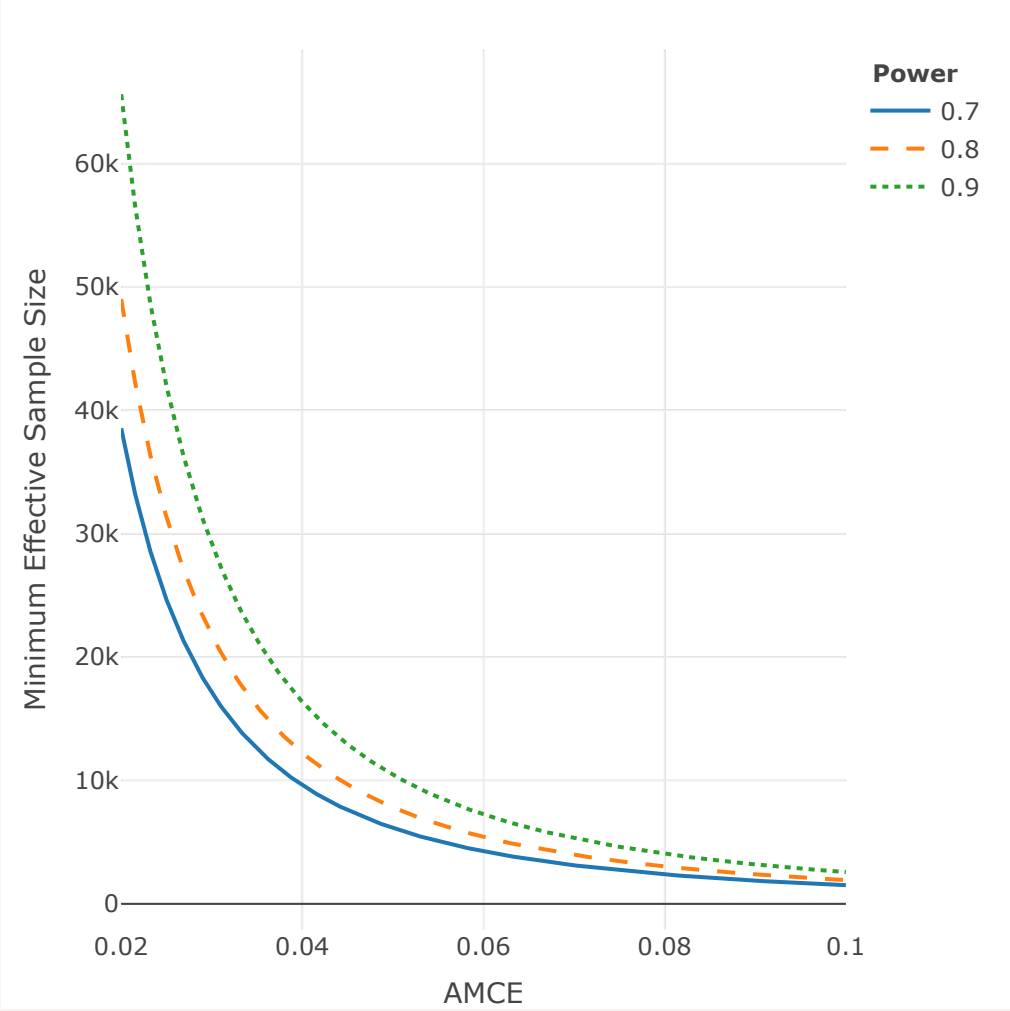


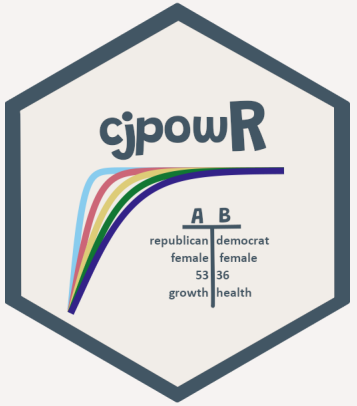
Showcase

Or maps...

Showcase

Or interactive graphs...





- If you are interested in **power for (conjoint/factorial) survey experiments**...

Showcase

Or or interactive maps...

Showcase

U can easily combine R code and text in so called Rmarkdown files to produce reproducible documents...E.g. this presentation, LaTeX `.pdf`s or even Word files.

R-Studio & (Git)Hub

Installation

Steps you should have done already:

1. Install **R**.
2. Install **R-Studio**.
3. Create a **GitHub** account. Take some care with the user-name if you want to keep this account throughout your career. You can't change it afterwards.
4. Install and set up **Git** (and optionally a desktop client).

R-Studio

R-Studio is an IDE (integrated development environment) for the R language:

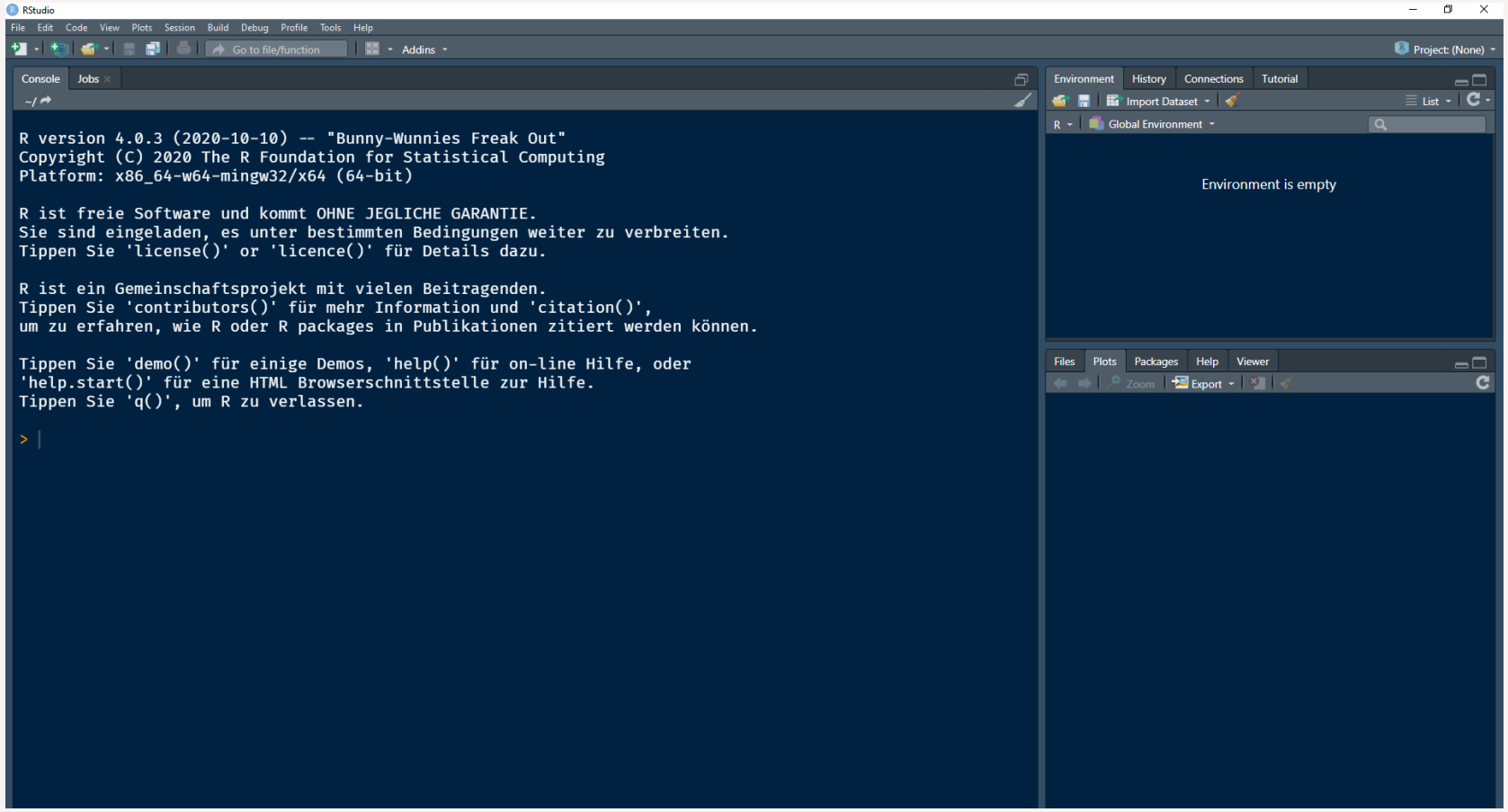
- comes with a console, code editor, tools for plotting, history, debugging and workspace
- open source
- pretty accessible/easy to use

Alternatives:

- Visual Studio Code (nice, if you work with multiple langs; my fav IDE)
- Alternatives are worse for package development, shiny platforms and some other R-specific stuff
- If your main language is R, RStudio is best

R-Studio

Lets take a tour...



A Small Detour: Two Types of "Scripts"

In this course, we will use two types of scripts to write our code:

1. Classic R-Scripts (`.R`): simple text file; comments are usually done like so: `# A comment.`
2. **Rmarkdown** files (`.rmd`): combines code and free text (+ figures and formulae)
    - Can be "knitted" to, e.g., .pdf, .html and Word
    - Makes your documents (e.g. a paper or a thesis) fully reproducible (more on this tomorrow!)
    - Nice for problem sets (hence, we will use it for this right from the beginning)

Rmarkdown

An Rmarkdown file consists of mainly three things:

1. **YAML header** (Yet Another Markdown Language). Specifies meta info (e.g. author, date, document format, etc.):

````
```r
---
title: "Untitled"
author: Markus
output: html_document
---
```
````

1. **Code chunks** sorrounded by ``` . You can execute each chunk individually.

2. Plain text formatted via Markdown, a markup language with very **easy syntax**.

R projects

- To keep our sanity when coding (and to produce something reproducible in the end), we want to keep all our data, analysis scripts, outputs (e.g. figures) etc. together.

Three approaches:

Bad

> Creating a folder in the explorer and dropping all files into it. Setting the working directory in the R script manually using, e.g., `setwd("C:/Users/XYZ/New Folder")`.

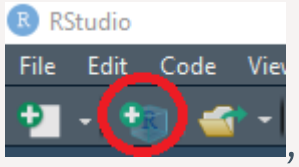
Just don't use `setwd()`. Ever.[1]

Why? Reproducibility.

[1] Yes, I am looking at you, Stata user, who loves to set working directories via `cd`.

R projects

Ok

> Clicking 
>
> , creating a local "R-Project".

This creates a folder with an `.Rproj` file. Whenever you open an R-project, a fresh instance of R starts and **the current working directory is set to the project directory.** You can then work with file paths relative to the project directory: E.g. `Figures/somepicture.png")`.

Good

> Creating a project and using version control.

This is where Git(Hub) comes in...

Why Git(Hub)?

Having multiple scripts inside your project called, e.g., `thesis_analysis_final_01_revised_2.R` is a nightmare.

- Using Git(Hub) improves your workflow:
    - helps with keeping track of the changes you do.
    - makes (code) collaboration with other researchers easy.
    - helps to make your research/code projects accessible/reproducible/open source.

Why Git(Hub)?

What's that thing called "Git"?

- It's a version control system:

$$\text{Git} \approx \text{MsOffice track changes and restore features} + \text{dropbox/google drive version history}$$

- But it's better, especially for any kind of projects that involve code
- You have to "commit" (i.e. save) actively, but that's a good thing (you could still have some sort of auto-save on you local machine)!

Why Git(Hub)?


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

GitHub

GitHub to the rescue:

- Built on top of Git
- A kind of online cloud service that makes working with Git easier
- again, no automated sync (but that's good!)
- Instead of some folder, your project lives in a remote **repository**

The remote repository is your upstream storage.

--> You can **clone** it from GitHub to create a local copy.

--> You can **fork** some repo (including those of other users); i.e. create a copy of the repo under "your repositories". You can then **clone** this forked repo to get a local copy.

Your first repo

1. Click **here** and create a new (Git)Hub repo. Call it "test", set it to private and initialize with readme.

2. In R-Studio, navigate to **File > New Project > Version Control > Git**.

3. Paste the Repository URL, chose a name and project path and **clone** the thing.

4. You will be asked to provide a personal access token. Generate it **here** using some name and check the "repo box".

5. In the files tab, open README.md. Also click on the Git tab.

Do this **now**!

4 Operations You Need to Know

1. **Stage** ("add")
    - Tells Git which files u want to make changes (edits, deletes, etc.) to in the repo (simplification); in R-Studio this boils down to "selecting" files/changes to files by checking them.
2. **Commit**
    - Git's way to "save" the changes you staged.
3. **Pull**
    - "downloads" all new changes/new commits from GitHub
4. **Push** (to origin)
    - "uploads" all commits to GitHub; to the origin, i.e. your upstream remote repo.

Commit

Make some changes to the `README.md` file, stage, commit and push.

To establish best practice, give your commit a meaningful name:

- In general, commit whenever you think you made a meaningful change
- Push a bit less often than you commit

Collaborate with (Git)Hub

- Using version control really comes to shine when collaborating.
- BUT: you are always collaborating. With your future self. Therefore, always use version control.
- To invite someone to a repo on GitHub, go to the repo `settings > manage access`.
- Your collaborator can then clone the repo and contribute commits, push them etc.

When Things Go Sideways: Merge Conflicts

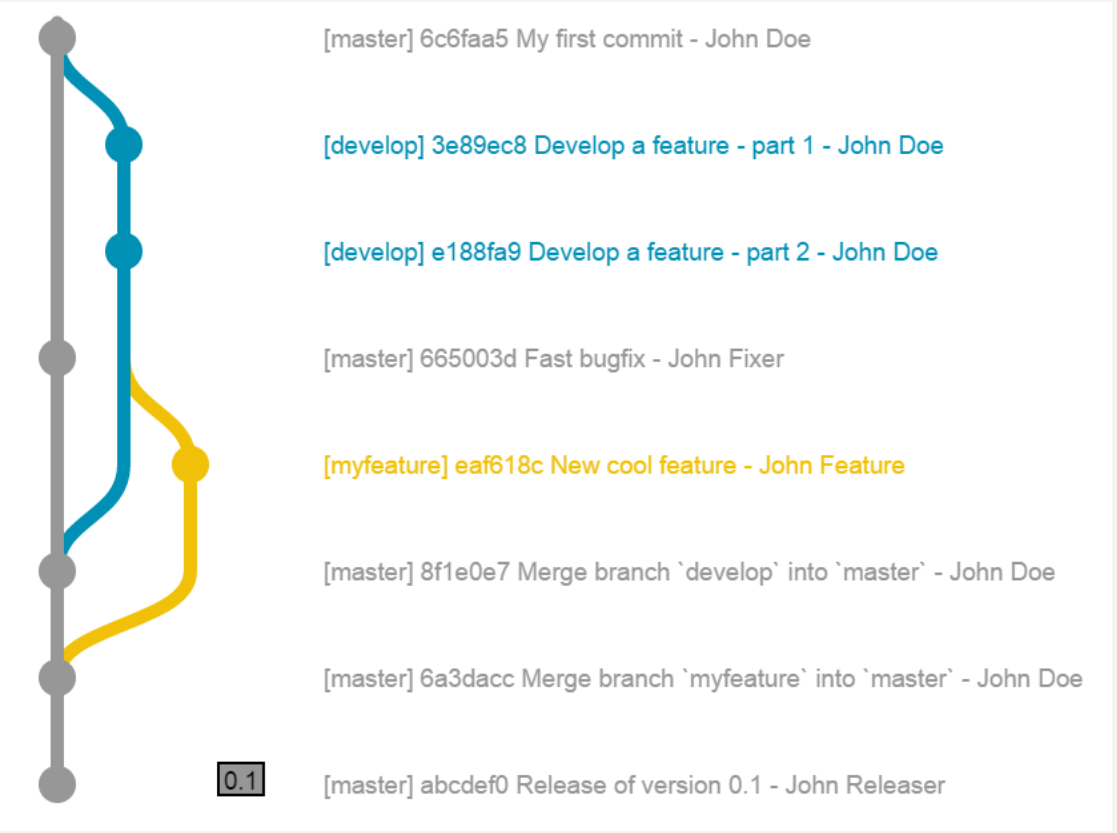
1. Go to your new repo. Edit the README.md manually in line **3**.
2. Commit some changes in the **same** line locally in R-Studio. Commit.
3. Pull (don't push).
4. Git:

When Things Go Sideways: Merge Conflicts

What do you do now?

- Well, you (maybe in exchange with your collaborator) decide!
- Solve the conflict manually, add, commit, push.

Branches

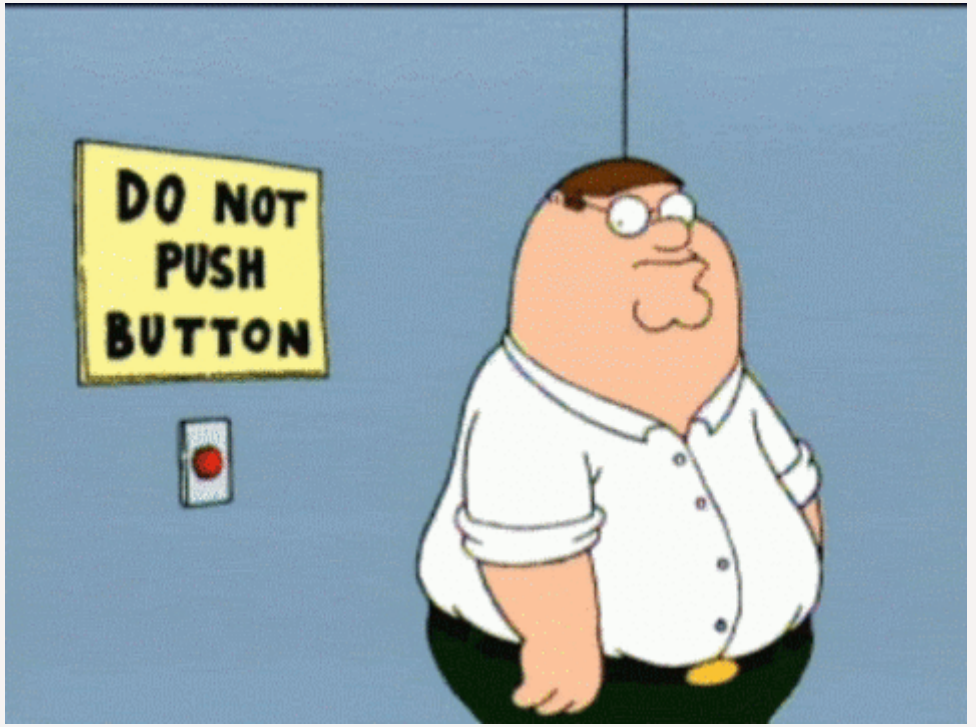

[master] 6c6faa5 My first commit - John Doe
[develop] 3e89ec8 Develop a feature - part 1 - John Doe
[develop] e188fa9 Develop a feature - part 2 - John Doe
[master] 665003d Fast bugfix - John Fixer
[myfeature] eaf618c New cool feature - John Feature
[master] 8f1e0e7 Merge branch `develop` into `master` - John Doe
[master] 6a3dacc Merge branch `myfeature` into `master` - John Doe
0.1
[master] abcdef0 Release of version 0.1 - John Releaser

Branches

- Branches allow you to develop/test some idea without touching the main version of your code.
- Useful for larger projects. You get a full copy of the repo and you can commit/pull/push all you want.
- If your idea turned out not to work. Just delete the branch.
- In R-Studio, they can be created via the little purple branch icon in the Git tab
- If you want to integrate the feature/idea into the main branch, issue a pull request (easiest way is via GitHub).

When Things Go Really, Really Wrong

- DON'T PUSH.
- If you did not, just clone a fresh instance of the repo.
- If you did, you can revert to an older version (but that's more work).



Workflow Summary

1. Create a repo or fork some existing repo
2. Invite collaborators
3. Clone it & create an R project
4. Edit code or make other changes
5. Stage, commit (with message), pull (esp. if u work with others/to avoid conflicts right away), push
6. Rinse and repeat steps 4-5.

Workflow Summary

Yes, the order really is stage, commit, pull, push.

Well, we got an intuition for it when we created a merge conflict. See this concise **stackoverflow** answer for a summary:

> It's better to commit first. Pulling without commiting may make your work overwritten. With a local commit, conflicts will be shown and prompted for manual merging when pulling, giving you a better control over your work.

Further Steps

- Fork the course repo and clone it via R-Studio (that's perhaps a bit hacky but easy and does the job)
- You will need this to conveniently access the course materials/problem sets.
- As the fork is your own copy of the repo, feel free to commit and push all you want.
- You could also use the **shell** to clone the repo or the, alternatively, GitHub Desktop
- For beginners, using a Git GUI (like GitHub Desktop) and R-Studio will cover well over 90% of the use cases imo

See, e.g., **here** for further reading.

Next Up: Base R & the Tidyverse