Before we start today...

- Create a Github account
- Share your account name with me at tomoyas@mit.edu
- If possible (or necessary), install Git in your computer (check `https://happygitwithr.com/install-git.html`)

Git for Social Scientists: Introduction to Version Control with Git

Tomoya Sasaki[1]

Massachusetts Institute of Technology

November 4th, 2022

[1]This slide deck is heavily inspired by the workshop materials by Suyeol Yun and Shiro Kuriwaki.

Introduction

Companies using Git

Airbnb
Allegro
Amazon Web Services - Labs
Amazon Web Services
Amazon
Ambientia Group Oy
Apple Inc.
arkency
AT&T Open Source

Companies using Github

Introduction

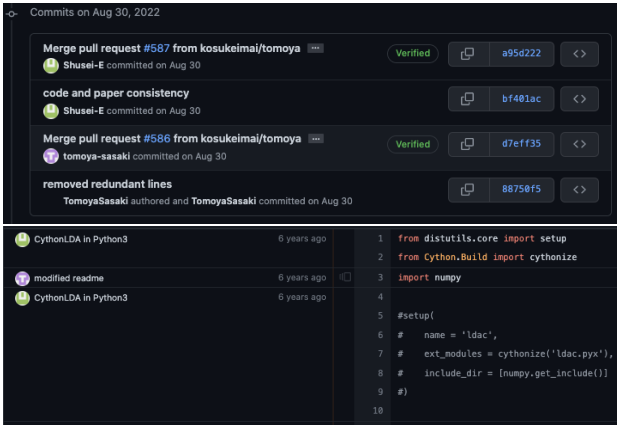
- Why Git? Why Github? Why version control?
- These are essential tools for programmers
- How about social social scientists?
- My opinion: Social scientists also benefit from version control with Git
  - Increase in collaborative projects
  - Demand for clean replication materials
  - Complex data manipulation/preprocessing/analysis
- "Code and Data for the Social Sciences: A Practitioner's Guide" by Gentzkow and Shapiro has a chapter dedicated for version control
- This workshop
  - Introduction to version control
  - Pros and cons of Git/Github
  - Brief introduction to these tools (how Git/Github works and how to use them)

What is version control?

- Version control: tracking and managing changes to file content
- Git: (the most popular) software for version control
- Github: service to host your git on the Internet (alternatives include GitLab, Bitbucket ...)
- Repository: unit of a version control project, your project folder with a subfolder named `.git`
  - Often simply called "repo"
  - `.git` folder in a repository tracks and stores every single change you make in the corresponding repository
- I focus on Git/Github because they are extremely popular than their alternatives

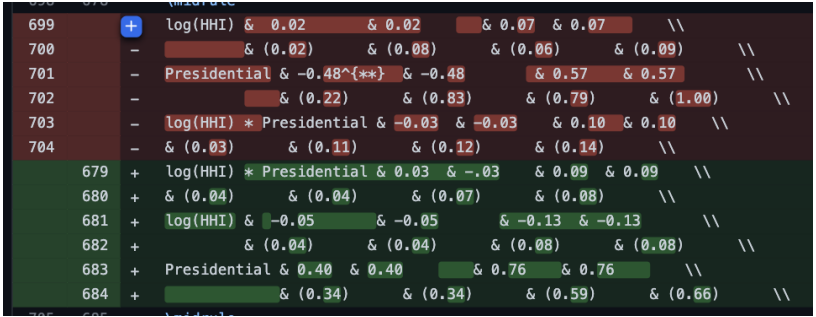
Benefit of Git/Github: Tracking who/how/when

- You can identify
  - who made changes
  - how they made the changes
  - when they made the changes
- You can check the entire history since you created a repository and move back to previous versions easily (undo changes)
- Github can visualize them nicely
- Useful when you
  - want to revert your (particular) changes
  - work on a collaborative project
- You don't need to keep
  - different versions of the same file: `clean_data_1104.R`, `clean_data_1020.R`
  - the same file edited by different people: `clean_data_tomoya.R`, `clean_data_adam.R`



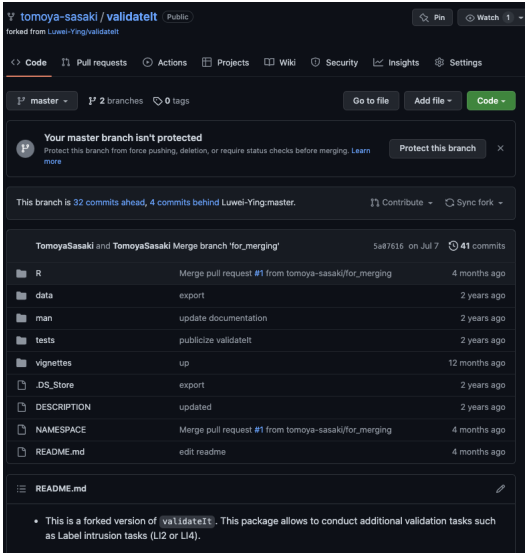
Benefit of Git/Github: Tracking who/how/when

- You can check how the results change when we try different specification
- Easy to track which part of the results changed



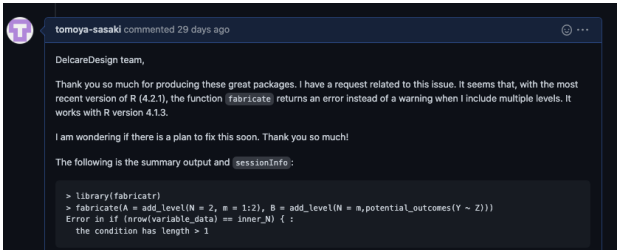
Other side benefits of Git/Github

- Hosting a customizable website (free, no ads, tons of templates)
- Contribute to software packages hosting on Github
- Tweak a package developed by someone else for your own purposes
- Send a request to package developer (often happens at "Issue")
- Nice integration with popular apps/websites such as RStudio and Overleaf



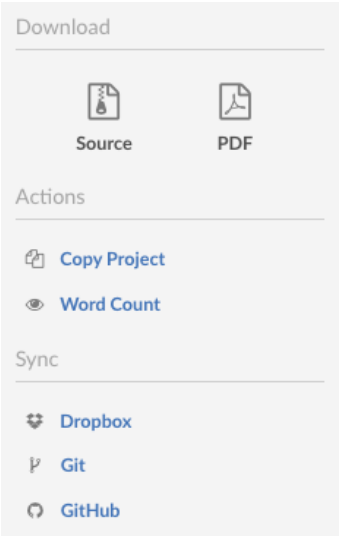
Other side benefits of Git/Github

- Hosting a customizable website (free, no ads, tons of templates)
- Contribute to software packages hosting on Github
- Tweak a package developed by someone else for your own purposes
- Send a request to package developer (often happens at "Issue")
- Nice integration with popular apps/websites such as RStudio and Overleaf



Other side benefits of Git/Github

- Hosting a customizable website (free, no ads, tons of templates)
- Contribute to software packages hosting on Github
- Tweak a package developed by someone else for your own purposes
- Send a request to package developer (often happens at "Issue")
- Nice integration with popular apps/websites such as RStudio and Overleaf



Limitations: not suitable for tracking large files

- Github imposes file size limit:
  - 25 MB per file limit (you can change this limit up to 100MB by changing setup)
  - 1GB per repository limit
- Remember that `.git` tracks and stores all the change you make in a repository

  $\rightsquigarrow$ if you store a huge file in the repository and let `.git` tracks its changes, the `.git` folder can grow quite huge

- Use `.gitignore` to specify files that Git should not track "ignore"

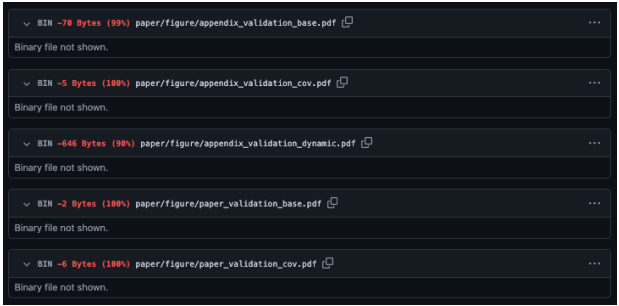
.gitignore

```
*.csv # ignore csv files
/data/ # ignore any file in data folder
```

- Include huge files as well as sensitive files that contain password, API key etc in `.gitignore`

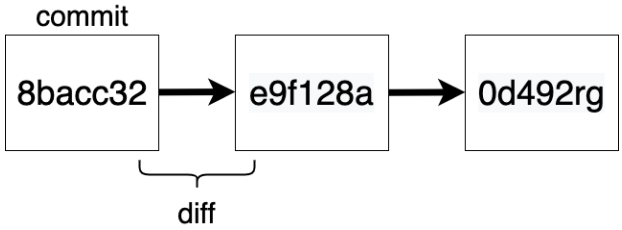
Limitations: not great if you want to track non-text files

- Git cannot track line by line changes for non-text files such as PDF, Microsoft Word/Excel/Powerpoint, JPG, ...
- Note that Git still tracks changes
- The value of Git/Github is limited
- In the right example, Git/Github recognizes the changes as the changes in file sizes
  $\rightsquigarrow$ even though you update a figure in PDF or PNG format, Git/Github might not recognize it unless the file size changes...



How Git tracks files

- Git tracks changes in files with "**commit**"
- Changes between commits are called "**diff**"
- Each commit is a snapshot of your repository at specific times
- Commit has a human-readable message and an (first few characters of) commit ID
- Commit is not automatic and you actively make a "commit" with a message
- Ideally commit should be atomic units of change that represent a specific idea
- You need to "**stage**" the files you want to add to commit (more later)



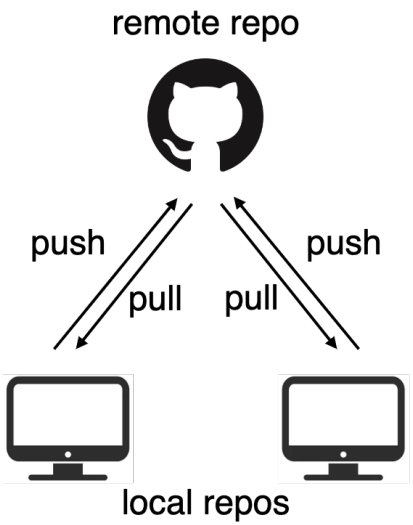
- 8bacc332: "test commit"
- e9f128a: "edit analysis code"
- 0d492rg: "add new analysis"

Github and more on repos

- If you just work on your local repos (and only on one computer), you don't need Github
- Repos: unit of a version control project, contains a folder with a subfolder named `.git`
  - Local repos: repos (folder) in your own computer, ordinary project folder with subfolder named `.git`
  - Remote repos: repos (folder) in a web hosting services such as Github
- Remote repos such as Github...
  - come with unique URL (`https://github.com/username/reponame.git`)
  - are basically a copy of your local repos
  - nicely visualize your commits and also serve as a backup of your codes (remember, not for data)
  - useful when you work on the same project with multiple computers
- You can choose remote repo to be public or private
- Github accounts let you create unlimited number of private repos with upto 3 collaborators
  unlimited number of public repos with unlimited number of collaborators

How to interact with remote repos on Github

- Interaction with remote repos: **pull** and **push**
- You **push** the commits you made to remote repos
- You **pull** new commits on remote repos
- Technically "pull" does **fetch** and **merge**



How to use Git/Github

- So far, we have covered some fundamental concepts around Git/Github
- But how do we initialize a repo and commit/pull/push in practice?
- Command line, Github desktop app etc
- This workshop focuses on RStudio
- I personally use command line

General (suggestive) initialization

1. Establish a connection between your computer and Github
2. Create a repo on Github (create a public one for now)
3. **Clone** your remote repo on Github to your computer (can be any location). This is your local repo.
4. Edit `.gitignore` to make sure that Git won't track any sensitive or huge file

- If you have any existing project that you want to track with Git, create a remote repo on Github, clone it to your computer (i.e., local repo), and move codes and folders into the local repo

General (suggestive) daily workflow

1. Edit, add, or delete files in your repo
2. Stage files you want to add to commit and make a commit when you make some changes that represent a specific idea or solve particular issues
    - Add lines to clean data
    - Add lines to visualize regression results
    - Delete lines that are irrelevant anymore
3. Push when you make a set of commits or when you call it a day
4. Pull any new commits in the remote repo if there are commits and pushes to the remote repo from other computers
5. Repeat the process above

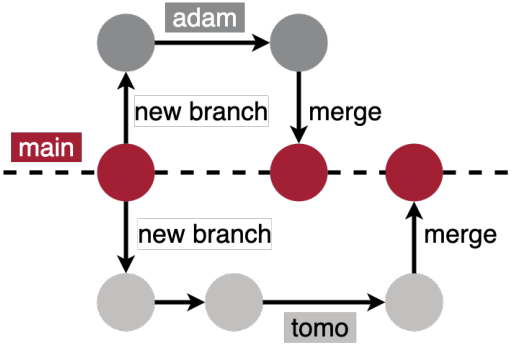
- Push at least once a day if you make any edit
- Let's try out with Github and RStudio

Setup authentication

- When we interact with a remote Git server, such as GitHub, we need to provide credentials
- Github provides two authentication methods, HTTPS and SSH
- I recommend HTTPS and will use HTTPS in this workshop
- Procedure
  1. Go to `https://github.com/settings/tokens` and click "Generate token"
  2. Decide the scope. Choose (at least) "repo", "user", and "workflow".
  3. Click "Generate token". This is the same as the password to interact with Git. You shouldn't show this to anyone.
  4. Copy the generated strings (PAT, personal access token)
  5. Open RStudio and run `gitcreds::gitcreds_set()` (Install the `gitcreds` package beforehand)
  6. Paste your PAT
  7. RStudio stores and remembers PAT for you

How Git/Github handle collarborative projects

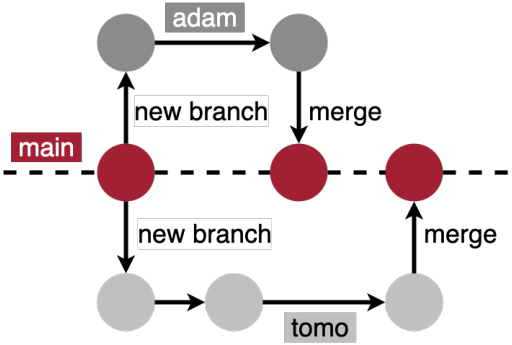
- Easy collaboration is a distinct feature of Git
- Key concepts: **branch** and **fork**
- Branches are parallel universe of a repository
- Main branch is the default branch and should contain stable version
- You contribute to a repo by creating a new branch in isolation from changes that other people are making to the repo
- Whenever you work on a collaborative project, you should always create a new branch



- Each circle represents "commit"
- In this example, Adam and Tomo work on the same project and each of them creates a branch

How Git/Github handle collarborative projects

- Once your work is done, open **pull request**, and **merge** your branch into the default branch
- Pull request is an opportunity to discuss or check (**review**) if the changes you make in your branch won't break the default branch
- By merging your branch into the default branch, any edit you make in your branch will be reflected in the default branch
- When both of them finish their work, they open a pull request and merge their branch into the main branch



- Each circle represent "commit"
- In this example, Adam and Tomo work on the same project and each of them creates a branch

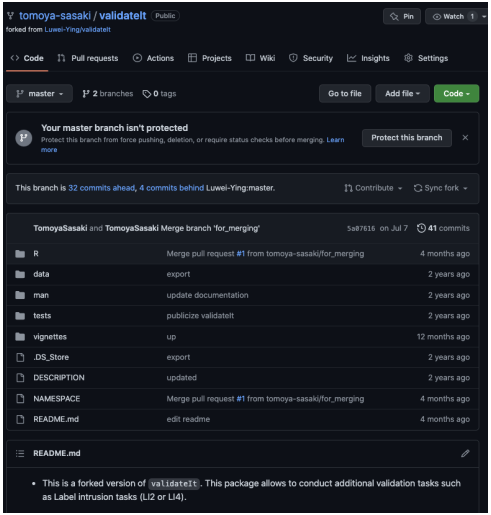
General (suggestive) daily workflow with multiple branches

1. Create a new branch (usually from the default branch)
2. In your branch...
    1. Edit, add, or delete files in your repo
    2. Stage files you want to add to commit and make a commit when you make some changes that represent a specific idea or solve particular issues
    3. Push when you make a set of commits or when you call it a day
    4. Repeat the process above
3. Create a pull request when you finish your project
4. Merge your branch into the default branch

- Let's try out with Github and RStudio

Working on someone else's repo

- Even if you have a public repository on Github, only those who have permission can directory push and merge to your repo
- What to do if you want to contribute to someone else's repo or borrow their idea and tailor for your own project?
- **Fork** their project to your Github account
  - Original: `https://github.com/xxx/reponame.git`
  - Forked repo: `https://github.com/yourusername/reponame.git`



Comparison and relationship to alternatives (Dropbox, Google Drive)

- Unlike Dropbox or Google Drive, Git/Github does not automatically track changes

  $\rightsquigarrow$ you need to take (small) action (mostly) when you
- However, you can make your project folder clean with commits and branches
- You can revert your changes easily and flexibly if you make frequent commits

  $\rightsquigarrow$ Dropbox and Google Drive only preserve once-in-a-day snapshots

- Putting Git local repos in Dropbox or Google Drive can be very tricky when your Dropbox/Google Drive account is synced to multiple computers
- Git does not like autosync (autoupdate) by Dropbox/Google Drive
- But you probably want to rely on these other services to store huge data that Git does not track
- In my setup, I put Git repos in Dropbox but sync them only with my main computer and the same Git repos in my other computers are located in non-Dropbox folders

What's next?

- Working with private repos
- (Gradually) Learning how to use git in command lines
- Probably you mess up once in a while. I still make mistakes sometimes
- You can fix them easily by reverting

Useful resources

- Happy Git and GitHub for the useR: https://happygitwithr.com/index.html
- Git Guide: https://github.com/git-guides
- git-vs-dropbox: https://michaelstepner.com/blog/git-vs-dropbox/