Open Science (OS) 2.0

An introduction to Git and Gitlab

Wassim Tarraf

3/14/2022

This presentation borrows liberally from Jennifer Bryan's electronic book: happy-git-with-r. I have learned tremendously from this and her other works (code, packages, tutorials, and courses). In the spirit of OS, many thanks to her and her collaborators for making these materials available for wide public consumption.

Context

- Purpose: Facilitate communication and collaboration; also useful in solo-analyst situations

- Version control system: Collaborative system to share information and track changes

  - *'diff's are the sets of differences over the evolution of a document (e.g. V2 vs. V1)*
  - *Learning about process from examining 'diff's*
  - *Git as an accumulation of 'diff's*
  - *'**commit**' is a reasoned approach to evolving through 'diff's. Everytime you choose to commit you are forced to provide a brief explanation of **why** you are committing to a version*
  - *The natural history of a project is documented through these acts of committments*
  - ***tag**s can be added to label critical designations (e.g. a version submitted to a journal)*

- Repository: a structured collection of files, that is continuously evolving

- Hosting services: homes for code; kind of like what Dropbox or OneDrive are for your files

  - *Private: only accessible by you and designates (different grades of designation)*
  - *Public: accessible by the world*
  - *local (on your machine) vs. host versions (shared and available across machines and users)*
  - *GitHub, or GitLab, etc…*

Digression into workflow

- Daily, weekly, monthly, yearly, …. routine by which you engage with work

- Coordinated framework for:

  - *Planning, organizing, and documenting scientific process*
  - *Establishing and fostering collaborations*
  - *Managing and sharing data*
  - *Analyzing data*
  - *Disseminating findings*
  - *Archiving process for replication*

- We all have it:

  - *For some it is linear and highly delineated*
  - *Others have a personalized non-linear way*
  - *For some others it is chaotic and non-streamlined*

Moving to an OS workflow

4 initial commitments:

1. Dedicate a folder to it
2. Make that folder a Git repository
3. If it the code involves the use of RStudio, make it an RStudio project
4. Commit to `commit` and not just save

Start solo and then propagate (to lab members, to collaborators, to audience (e.g. students), to public)

Decision Points

- Creating own workflow or using existing Workflow Management Systems (some recommendations)
- Choosing a Data Repository (not covered)
- Deciding on a Source Code Repository (some recommendations)
- Choosing a system to "Package, Access, and Execute Data and Code" (some recommendations)
- Choosing a Document Repository (free or fee for service) (limited recommendations)
- Licensing and Privacy (not covered)

What does an open science workflow get you?

- Primary Benefit is facilitating replication and strengthening the evidence base
  - *Internally: Streamlined analytic process*
  - *Externally: Improved collaboration*
- Efficiencies:
  - *Better framework for fixing and recovering from errors*
  - *Enhanced throughput*
  - *Use of past processes to inform future work*
  - *More natural evolution of scientific product*
  - *Inheritance*

Replication

- Workflow effectiveness => enables replication
- Be planful starting today
- Universal concern with replication in scientific fields
- Easy metric for success
- Use existing gauge – your "manuscript" is ready when it is ready for public assessment
- Your work is replicable if your scientific process is ready for public view

Replication is complicated!

Ask yourself: **Can someone else using my project files, comprehend my process, understand the reasoning for my decisions, and reproduce my findings?**

Answer is in Documentation:
- Detailing of process - nothing left to memory
- Explicit choice of tools that facilitate public documentation of scientific process

- Protection against document leakages (loss, corruption, obsolescence); version control

Some Criteria to consider (Scott Long)

- How simple is it to use?
  - *Critical in the beginning*
- Is it suitable for your personal needs?
- Does it enhance current workflow and is it sustainable?
- Is it sustainable as a "longer-term" solution?
- Can it be scaled to expected growth (multiple projects, lab needs, collaborations)?
- Does it contribute to standardizing critical production elements?
- Does it help with automating repetitive tasks?

Complications of collaborations

- Collaboration can be a hazard for breakages in workflow
- Unless system includes:
  - *Clear role definitions*
  - *Standards for interacting and feeding into the established system*
  - *Mechanisms for coordination*
  - *Enforcement rules*

Systemic challenges

- Individual research needs
- Incentive structures not yet established
  - *Rewards for "openness" not yet fully recognized*
- Time costs
  - *To set up the system*
  - *To be productive within the system*
- Other systemic constraints (e.g. data restrictions)

Changing a workflow

1. Slowly, systematically, thoughtfully
2. Master a few tricks at a time and integrate over time
3. Don't think it or attempt to change it under deadline
4. Many viable workflows:
    - *Find one that might work (borrow from other efficient users) with your style and personality*
    - *Make it your own and instill it in your lab members*
    - *Be flexible to change; be open to having graduate students, post docs, and collaborators show you new ways*

Steps

1. Create a dedicated directory

2. Make it a Git repo

3. Commit changes (the equivalent of saving; think of this as a solution of your current naming nightmares **“copy of copy of copy of code created June 10 1998_upgraded June 18th 2003_with_WT_&collaborator_edits”**

4. Push commits to host – make it visible to others

5. Address potential conflicts: merge

6. Think of Git as providing Google doc advantages; with some complications

Additional features

Issues:

- Tracking system for bugs or problems that show up along the way
- System for communicating and embedding discussions through code build up
- Documentation for how problems/disagreements get resolved over the course of project development

Pull Requests:

- Branching
- Allows for simultaneous differed approach to the same problem
- Segmentation/division of labor
- Space for tackling potential solutions for bugs
- Ultimately can be merged to improve the process and expected outcomes

Register

GitHub

GitLab

- Start with a free account and then explore offers and upgrades
  - *Potentially needed based on structure of collaborations, requirements for coordination across repos and users*
  - *There are paid options (pricing depends on needs; most project work requires no pay). Decision could be done later*

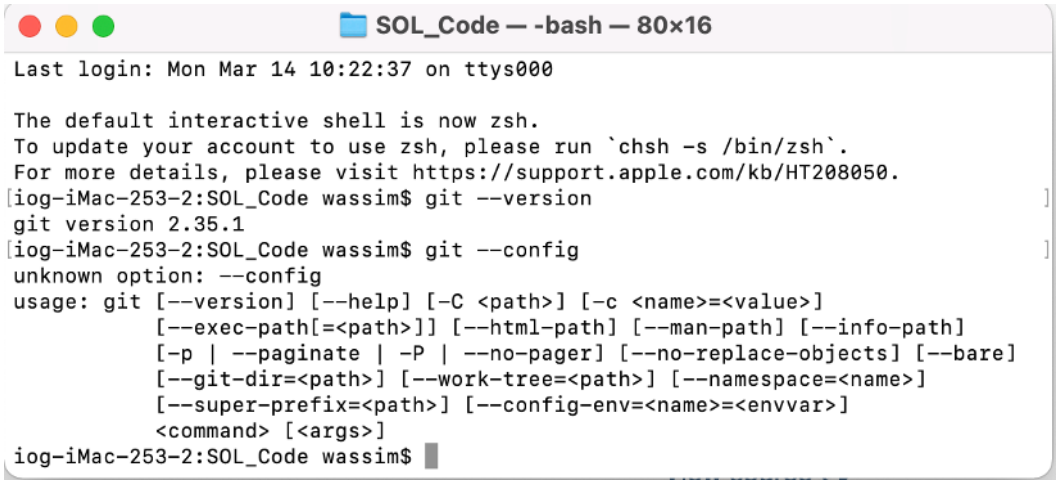
R and RStudio

- Download  and 
- Stay as current as possible with version release
- Ensure that packages are upgraded regularly
  - *Potential for conflict when upgrading, also useful to maintain older version of packages and use depending on need*

For code guidance on how to sign up to GitHub, install Git, configure Git with RStudio, and verify the setting see this link

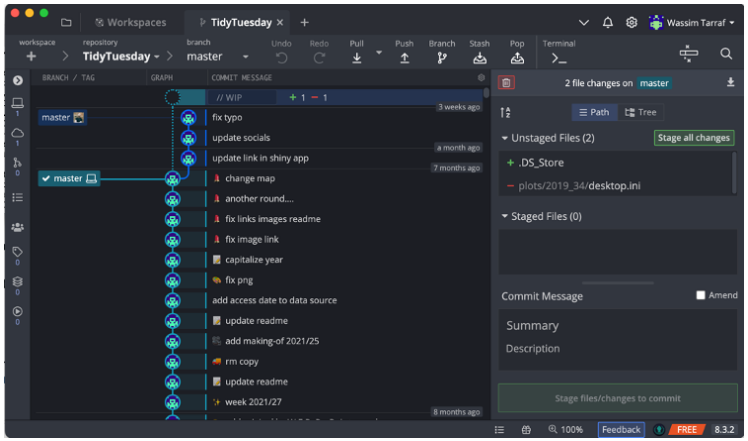
Working with Git

1. Work with Git directly from Terminal



2. Use a Git client

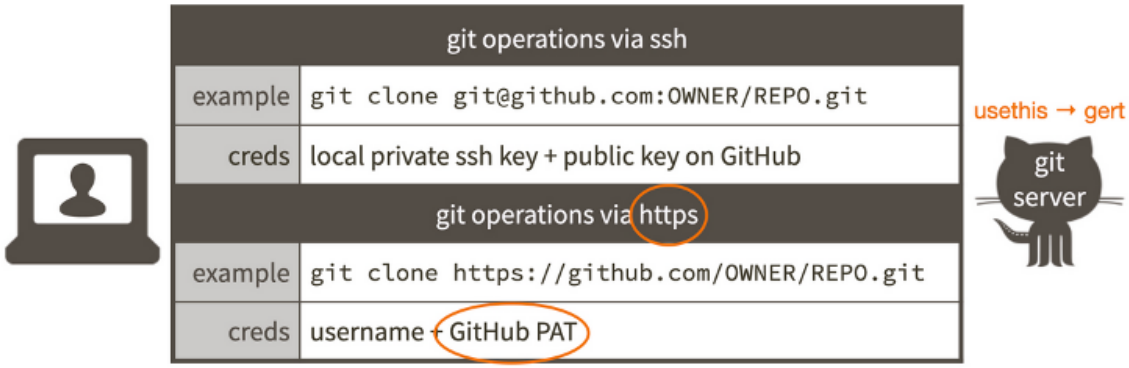
- e.g. GitKraken



Establish credentials

Required to communicate between your machine and the host; Communicate (i.e. the URL that your remote server is configured with) through (not an either or decision) two methods for establishing authentication (but mixed use means that you have to establish both credentials).

HTTPS (recommended for starters): Uses a personal access token (PAT), easy to use PAT should be stored in a secure and accessible system (problematic on Linux; can be configured to cache rather than store credentials)



In R:

- `usethis`, `gitcreds` packages for accessing and storing credentials

- `usethis::create_github_token()`
- `gitcreds::gitcreds_set()`
- looks like: https://github.com//.git
- SSH (potentially more secure, but slightly more difficult to set-up):

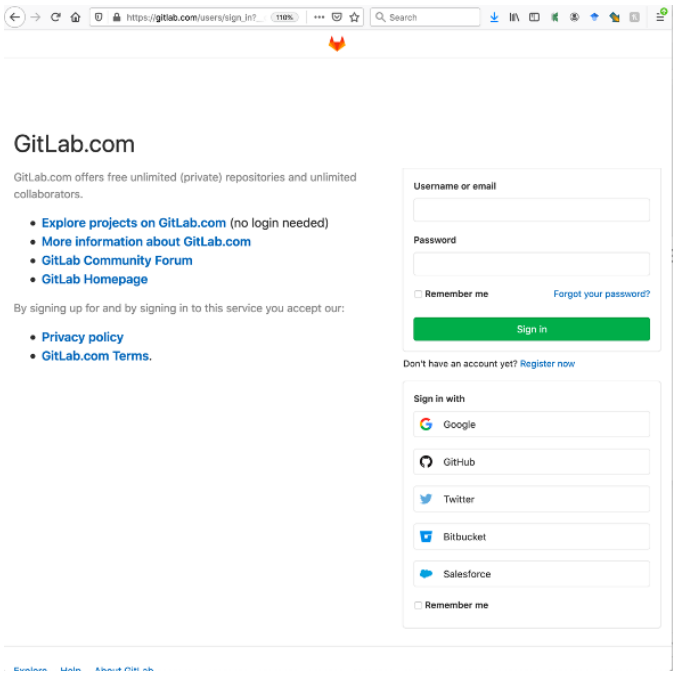
In Terminal:

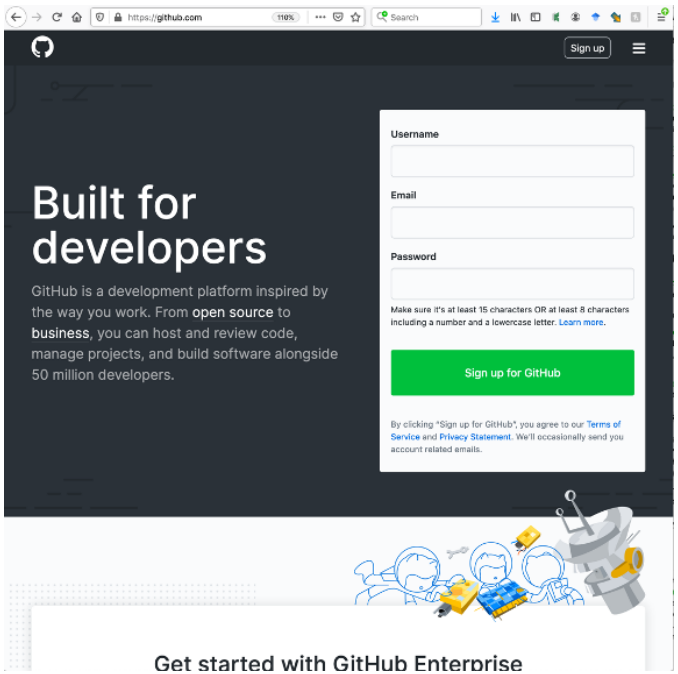
- `ssh`
- looks like: git@github.com:/.git

GitHub or GitLab

What are these?

- Platforms for hosting (mostly)
- Software based on Git
  - *distributed version control – peer-to-peer – each user has local copy and access to the full history of change*
  - *Mostly used in open source projects*
  - *Offer functionalities for code management (branching, merging, forking, cloning, etc..)*





Connect to GitHub

1. Establish a working directory

```
mkdir gitprojects
cd gitprojects
```

2. Clone a repo

```
git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git
```

3. Change directory to set to the cloned directory

```
cd YOUR-REPOSITORY
```

4. List all the files in the repo

```
ls
```

5. Display a readme file

```
head README.md
```

6. Get information on connection to GitHub

```
git remote show origin
```

7. Create a file in the repo

```
echo “Add text to a file on local” >> README>md
```

8. Check the status of the working directory

```
git status
```

9. Add a file to the staging area; include the file on the list of the next commit (i.e. changes to be pushed from the local to the host)

```
git add README.md
```

10. Commit the changes (additions) that are staged for moving to host (add a message explaining the changes commiteed)

```
git commit –m “Add a readme file"
```

11. Push the committed changes up to the host

```
git push
```

Given that you have appropriate credentials and that the connection between the local and the host could be established, the changes will be pushed and will become available through the host.

Connect RStudio to Git & GitHub

This is an optional step

1. Create a repo on GitHub

    - *Copy the https URL to clone*

2. Within Rstudio

    - *File -> New Project -> Version Control -> Git*
    - *Paste the URL copied from GitHub, keep repo name the same as the one created on GitHub*
    - *Ensure that the repo is saved to a location of interest*
    - *Check "Open in a new session" to ensure that a separate project session, specific to this project, is initiated.*
    - *Create the project*
    - *A new Rstudio session will be initiated and the readme.md file as well as a .gitignore file will be visible in the project files*

3. Open the readme.md file within Rstudio, make changes by adding some text, and save the file

4. Now you are ready to start the process of moving from the local to the host

    - *Click the Git tab in the upper left pane*
    - *Check the "Staged" box for the readme.md file where changes were made*
    - *Click "Commit" and type a message in the "Commit message" box that describes changes*
    - *Click "Commit"*
    - *Click push to move changes to host*

5. Go back to GitHub on the browser to examine whether changes were moved to host

    - *Refresh browsers; committed and pushed changes to the readme.md files should appear on the host*

6. Process can be applied to any file type not specifically excluded through .gitignore

Working with Git from Terminal

A collection of useful Git commands for starters as listed in Jenny Bryan's Happy Git and GitHub for the useR, Chapter 21.

Clone a new repo to local from Github host:

```
git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git
```

Check if clone worked:

```
git remote --verbose
```

Make local changes, stage them and commit to next step of pushing to host:

```
git add filename.txt
git commit --message "First commit"
```

Check Git status:

```
git status
git log
git log --oneline
```

Compare versions:

```
git diff
```

Add a remote repo to a local (existing) repo:

```
git remote add origin https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git
git remote --verbose
git remote show origin
```

Push local `main` to GitHub `main` and have local `main` track `main` on GitHub:

```
git push --set-upstream origin main
# shorter form
git push -u origin main
# you only need to set upstream tracking once!
```

Push local commits to Github:

```
git push
# the above usually implies (and certainly does in our tutorial)
git push origin main
# git push [remote-name] [branch-name]
```

Pull commits made on Github to local repo:

```
git pull
```

Pull commits and avoid potential merge conflict (un-reconciled diffs):

```
git pull --ff-only
```

Update the local repo to the parity with remote host

```
git fetch
```

Switch to a branch (potentially with an alternative version of code)

```
git checkout [branch-name]
```

Check remote and branch tracking

```
git remote -v
git remote show origin
git branch -vv
```

Next steps

Branching:

- Moves work away from main version of project to a subsidiary (parallel) stream where development can move without disrupting the main work.
- Branches usually include experimental work to evolve (or not) the main work with least disruptions. Used for relatively small deviations to accomplish potentially non-standard developmental tasks.
- Expectations is that work would be abandoned or reconciled with main version relatively quickly.

```
git branch dev
git checkout dev
```

Stopping work and switching out and then back can be easily achieved

```
# Option 1
git stash
# Option 2
git commit
# Switch out
git checkout main
# Switch back in
```

```
git chekout dev
# Wipe out the commit from above and bring back to the parent of current commit (i.e. most
recent commit)
git reset HEAD^
```

When done we can merge with main

```
git checkout main
git merge dev
```

In case of problem with merge. Reconciliation can be abandoned fairly quickly

```
git merge --abort
```

Remotes:

- One project can have multiple remotes on the host.
- As with switching between branches we can switch between remotes.
- The same functionalities apply across remotes including, add, fetch, merge, pull, push, etc…

YES IT MIGHT BE COMPLICATED *Hick-ups, failures, and problems are inevitable and frustrating. The silver lining is that with version control, unlike with real life, you can recover the recent past, and even reach into the depth of the long past.*

AND you can restart. Also, unlike real life you can burn it all down and like a pheonix emerging from the aches rebuild better for a brighter future.

For more on Git and Github read Jenny Bryan's full electronic book, clone the source or fork it and dare to go beyooooo…ooond