Open Science (OS) 2.0

An introduction to Git and Gitlab

Wassim Tarraf

2023-03-13

Credit where due

This presentation borrows liberally from Jennifer Bryan's electronic book: happy-git-with-r.

I have learned tremendously from this and therr other works (code, packages, tutorials, and courses). Many thanks to them and the collaborators for making these materials available for wide public consumption.

The presentation also borrows heavily from Greg Wilson's *Introduction to Git for Data Science*. For a git version of the original content see this.

The above resources, in the spirit of open science, are provided by their authors under the terms of the Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC 4.0) license. See here for licensing information.

The presentation also integrates several ideas from Scott Long's Principles of Workflow in Data Analysis. For Stata users, Long's Book titled The Workflow of Data Analysis Using Stata is also a great resource.

The presentation was written in Quarto which is a open-source software for publishing that uses a converter system that can process multiple text and other formats (for our purposes markdown) called Pandoc using (RStudio Desktop]https://posit.co/download/rstudio-desktop/), resources that are freely available to users.

Plan

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…*

Workflow

Digression into workflow (see Scott Long)

- Daily, weekly, monthly, yearly, …. routine by which you engage with work
- A process: Plan, organize, and document
  - *Establish and foster collaborations*
  - *Manage and share data*
  - *Analyze data*
  - *Disseminate findings*
  - *Archive for replication*
- It varies:
  - *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 (see this)

- 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 rigor and reproducibility** and **strengthening the evidence base**
  - *Internally (requires committment): Streamlined analytic process*
  - *Externally (might take time): Improved collaborations*
- Efficiencies:
  - *Better for fixing and recovering from errors*
  - *Enhanced throughput*
  - *Use of past processes to inform future work*
  - *More natural evolution of scientific products*

Rigor and Reproducibility

- Focus on replication (Universal concern with replication in scientific fields)

- Be planful
- Use existing gauge – similar to “study is ready when ready for public assessment”; “process is replicable if it is ready for public view”

<u>Replication is complicated!</u>

<u>Question:</u>

Can someone else using my project files (a) understand my process, (b) see the reasoning for my decisions, and (c) reproduce my results?

<u>Answer:</u>

- Documentation!
  - *Detail the process - do not rely on memory*
  - *Adopt tools that facilitate public documentation of scientific processes*
  - *Protect against document leakages (loss, corruption, obsolescence); version control*

What to Expect?

Some criteria to consider (see Scott Long)

- How simple is the process to set up and use?
  - *Critical in the beginning*
  - *Could be a steep learning curve*
- Is it suitable for your personal/lab 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 scientific production elements (e.g. uniformity in documentation, or data coding)?
- Does it help with automating repetitive tasks (e.g. regenerating merged data sets, or recoding common elements of data)?

Complications of collaborations

- Collaboration can lead to breakages in workflow, unless system includes:
  - *Clear role definitions*
  - *Standards for interacting and feeding into the established process*
  - *Mechanisms for coordination*
  - *Enforcement rules*
- VC systems have the potential to faciliate this

Systemic challenges see this

- Individual research needs
- Incentive structures not yet established
  - *Rewards for "openness" not yet fully recognized*
- Time and resources
  - *To set up the system*
  - *To be productive within the system*
  - *To teach and propagate th system*

- Other systemic constraints (e.g. data restrictions)

How to?

Establishing/evolving a workflow (see Scott Long)

1. "Slowly, systematically, and thougtfully": a few adoptions mastered and integrated overtime

2. How:

    - *Find (develop) a process that works for you (borrow from other efficient users). Upgrade it over time.*
    - *Make it your own and instill it in your lab members*
    - *Adopt a change mentality; be open to having graduate students, post docs, and collaborators show you new ways*

Steps to start

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 made later.*

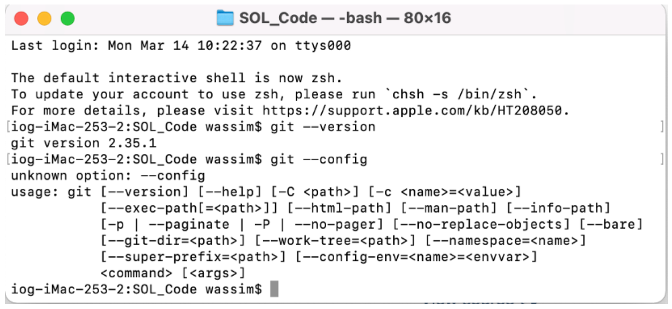
R and RStudio

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

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

Working with Git

1. Work with Git directly from Terminal



```
Last login: Mon Mar 14 10:22:37 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
iog-iMac-253-2:SOL_Code wassim$ git --version
git version 2.35.1
iog-iMac-253-2:SOL_Code wassim$ git --config
unknown option: --config
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]
iog-iMac-253-2:SOL_Code wassim$ 
```

2. Use a Git client (e.g *what I use* GitKraken)

Establish credentials

Required to communicate between your machine and the host

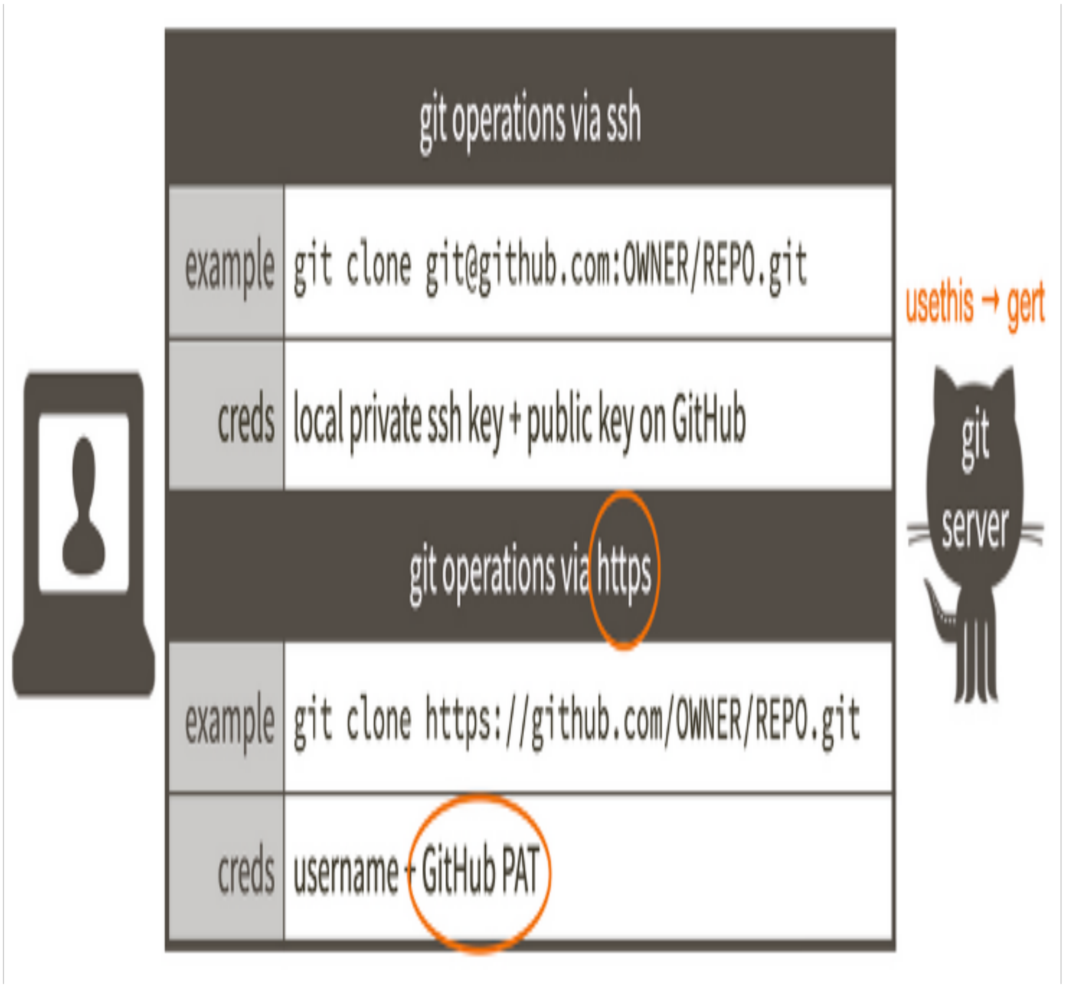
Communicate (i.e. the URL that your remote server is configured with) through (not an either or decision) two methods for establishing authentication (mixed use means that you have to establish both credentials).

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)

- looks like: https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git

SSH (more advanced): potentially more secure, but slightly more difficult to set-up. Allows for encrypted data communication using a pair of private and public keys

- looks like: git@github.com:YOUR-USERNAME/YOUR-REPOSITORY.git


git operations via ssh
example
git clone git@github.com:OWNER/REPO.git
creds
local private ssh key + public key on GitHub
git operations via https
example
git clone https://github.com/OWNER/REPO.git
creds
username + GitHub PAT
usethis → gert
git server

In R:

- `usethis`, `gitcreds` packages for accessing and storing credentials
- `usethis::create_github_token()`
- `gitcreds::gitcreds_set()`

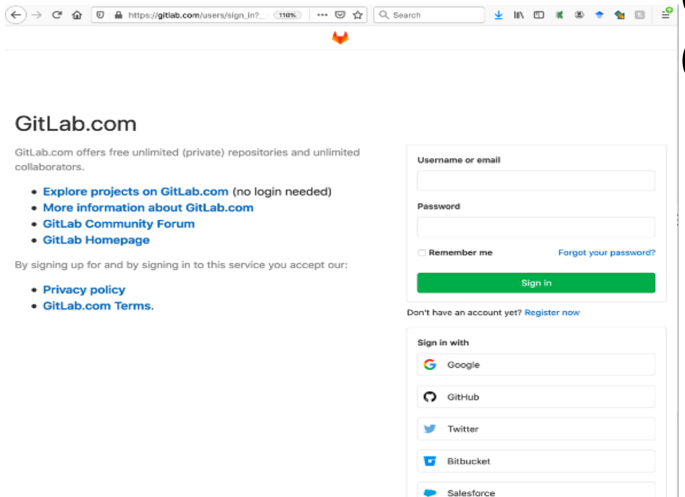
In Terminal:

- `ssh` see this for a Github example

GitHub or GitLab

What are these?

- Platforms for hosting (mostly) projects. This means you don't need to create a system (not your own server) for generating, maintaining, and connecting resources to the internet, this is offered by the service
- Software based on Git
  - *Distributed version control – peer-to-peer – each user has local copy and access to a remote copy and the full history of change*
  - *Assets for open source projects*
  - *In the context of what we are talking about today, they offer functionalities for project management, documentation, and code maintenance (sharing, branching, merging, forking, cloning, etc..)*



GitLab.com

GitLab.com offers free unlimited (private) repositories and unlimited collaborators.

- Explore projects on GitLab.com (no login needed)
- More information about GitLab.com
- GitLab Community Forum
- GitLab Homepage

By signing up for and by signing in to this service you accept our:

- Privacy policy
- GitLab.com Terms.

Username or email

Password

Remember me Forgot your password?

Sign in

Don't have an account yet? Register now

Sign in with

Google

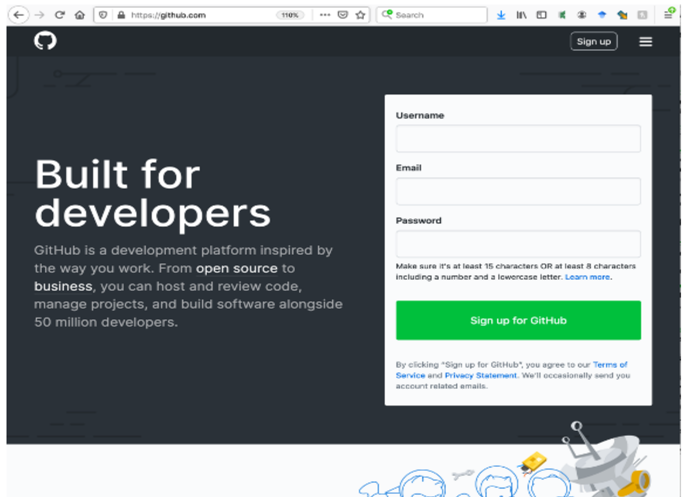
GitHub

Twitter

Bitbucket

Salesforce

Connect to GitHub:



Built for developers

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 50 million developers.

Username

Email

Password

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our Terms of Service and Privacy Statement. We'll occasionally send you account related emails.

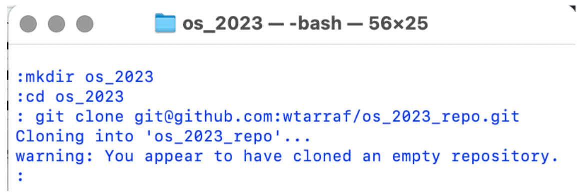
Demonstration through terminal

- Establish a working directory

```
# Create a directory called os_2023
  mkdir os_2023

# Set working directory as os_2023
  cd os_2023
```

- Clone a repo

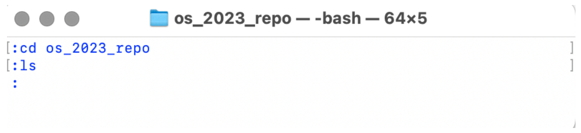
```
# General format
 git clone https://github.com/YOUR-USERNAME
/YOUR-REPOSITORY.git
# Example (not that this is in https
format)
 git clone https://github.com/wtarraf
/os_2023_repo.git
# Alternative with SSH
 git clone
git\@github.com:wtarraf/os_2023_repo.git
```



```
:mkdir os_2023
:cd os_2023
: git clone git@github.com:wtarraf/os_2023_repo.git
Cloning into 'os_2023_repo'...
warning: You appear to have cloned an empty repository.
:
```

- Change directory to set to the cloned directory

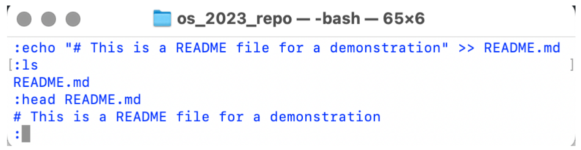
```
# General format
cd YOUR-REPOSITORY
# Example
cd os_2023_repo
# List all the files in the repo
ls
```



```
:cd os_2023_repo
:ls
:
```

- Display a readme file

```
# First create the README file
echo "# This is a README file for a
demonstration" >> README>md
# Note the change in what is in the
directory
ls
# Display the top of the README file
head README.md
```

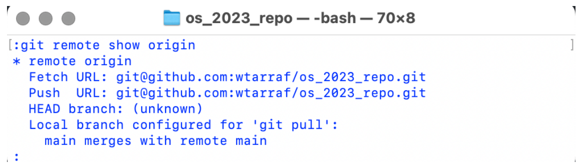


os_2023_repo — -bash — 65×6

```
:echo "# This is a README file for a demonstration" >> README.md
:ls
README.md
:head README.md
# This is a README file for a demonstration
:
```

- Get information on connection to GitHub

```
git remote show origin
```

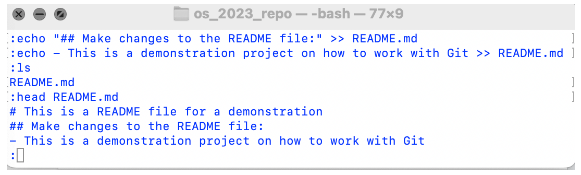


os_2023_repo — -bash — 70×8

```
:git remote show origin
* remote origin
  Fetch URL: git@github.com:wtarraf/os_2023_repo.git
  Push  URL: git@github.com:wtarraf/os_2023_repo.git
  HEAD branch: (unknown)
  Local branch configured for 'git pull':
    main merges with remote main
:
```

- Makes changes to file on the local

```
# Add two lines
echo "## Make changes to the README file:"
>> README>md
echo – This is a demonstration project on
how to work with Git >> README>md
# Check that the file is still there
ls
# Examine content of file
head README.md
```

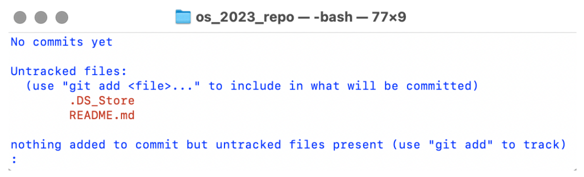


os_2023_repo — -bash — 77×9

```
:echo "## Make changes to the README file:" >> README.md
:echo – This is a demonstration project on how to work with Git >> README.md
:ls
README.md
:head README.md
# This is a README file for a demonstration
## Make changes to the README file:
– This is a demonstration project on how to work with Git
:
```

- Check the status of the working directory

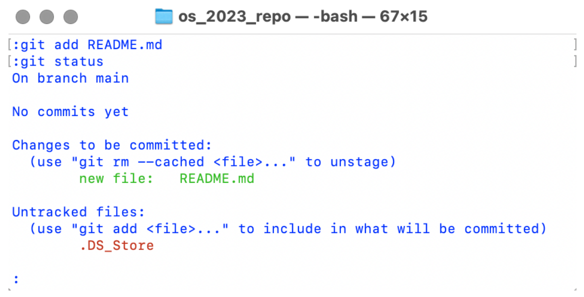
```
git status
```



```
No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .DS_Store
        README.md

nothing added to commit but untracked files present (use "git add" to track)
:
```

- 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)

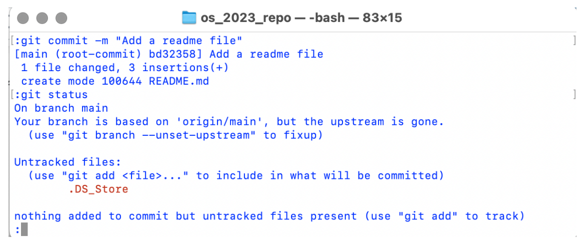
```
git add README.md
```



```
:git add README.md
:git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .DS_Store

:
```

- Commit the changes (additions) that are staged for moving to host (add a message explaining the changes committed)

```
# Commit the the README file (stage it for
pushing to the remote repo), adding a
message about what was done
git commit –m "Add a readme file"
# Check the status
git status
```

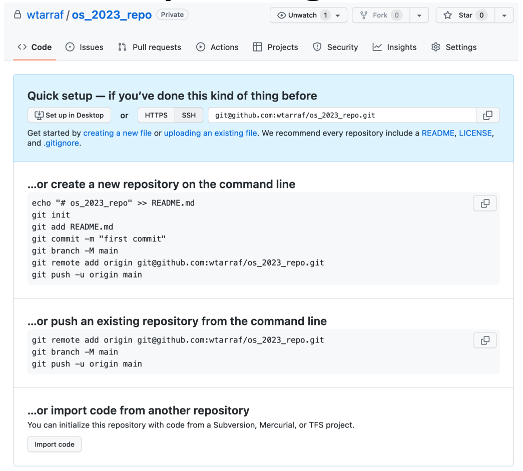


```
:git commit -m "Add a readme file"
[main (root-commit) bd32358] Add a readme file
 1 file changed, 3 insertions(+)
 create mode 100644 README.md
:git status
On branch main
Your branch is based on 'origin/main', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .DS_Store

nothing added to commit but untracked files present (use "git add" to track)
:
```

- Push the committed changes up to the host

```
git push
```

Before pushing



Quick setup — if you've done this kind of thing before

git@github.com:wtarraf/os_2023_repo.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

…or create a new repository on the command line

```
echo "# os_2023_repo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:wtarraf/os_2023_repo.git
git push -u origin main
```

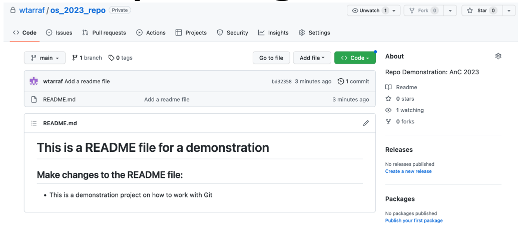
…or push an existing repository from the command line

```
git remote add origin git@github.com:wtarraf/os_2023_repo.git
git branch -M main
git push -u origin main
```

…or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

After pushing



Repo Demonstration: AnC 2023

wtarraf Add a readme file bd32358 3 minutes ago 1 commit

README.md Add a readme file 3 minutes ago

README.md

This is a README file for a demonstration

Make changes to the README file:

- This is a demonstration project on how to work with Git

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.

If you make change to the remote, or if your analysts/collaborators make changes and push them to the remote, then you can reconcile what you have on your local with that by doing this additional step.

- Pull changes from the host to the local

```
git pull
```

Connect RStudio to Git & GitHub

This is an optional step

1. Create a repo on GitHub
    - *Copy the https URL to clone (or use SSH)*
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

Quick Demo

More to explore

More on 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.

Branching:

- Moves work away from main version of project to a different stream where development can happen outside the main stream of coding.
- Branches usually include experimental work to evolve (or not) the main work with least disruptions.
- 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
# reset commit and bring back to parent
git reset HEAD^
```

When done we can merge with main

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

In case of problem with merge you can abandon.

```
git merge --abort
```

After merging we can also get rid of the development branch

```
# Delete on the local
  git branch --delete main/dev
# Delete on the remote
  git push origin --delete dev
```

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…

Final note

Try

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