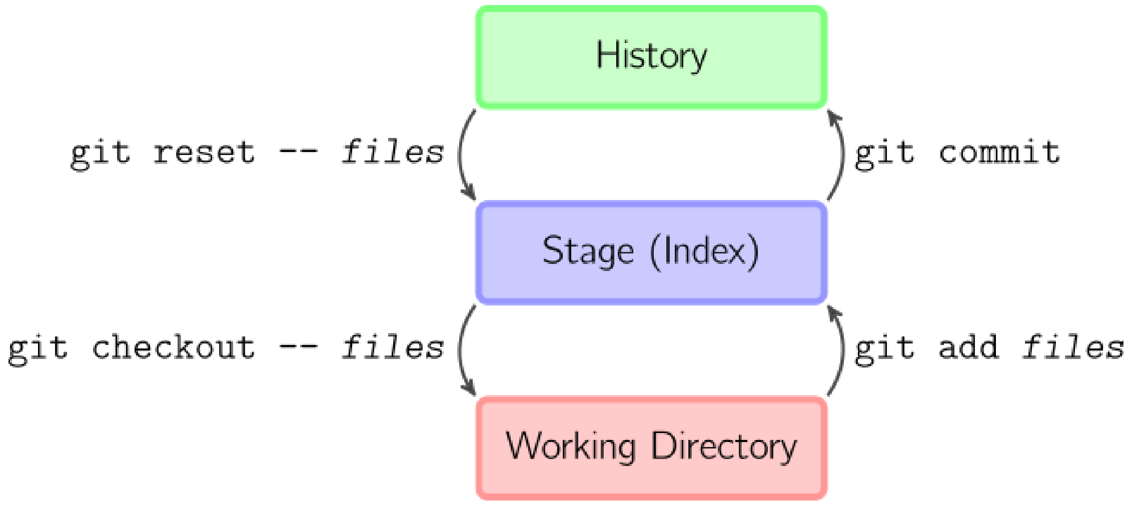
Git & GitHub

Thanks to professors Dickens, Ressler, and Gerard

Basic Git

- A **repository** (or repo, for short) is a collection of files (in a folder and its subfolders) that are together under version control. In data analysis, each repository is typically one project (like a data analysis, a homework, or a collection of code that performs a similar task).
- The way git works (graphic from Mark Lodato):



- **Working Directory**: To git, this means the current versions of the files. Changes to files that you haven't recorded only exist in the working directory and are not yet saved in the history.
- **Stage**: Files that are scheduled to be committed to the history, but not yet committed. Only files in the stage will be committed to the history.
- **History**: The timeline of snapshots of files. You commit a file to the history and then, even if you modify it later, you can always go back to that same file version.

- We'll focus on the right-hand-side of this diagram where your workflow is typically:

  1. Modify files in your working directory until you want a snapshot.
  2. Add these modified files to the staging area.
  3. Commit staged files to history, where they will be kept forever.

- The left-hand side of the diagram is used when you want to undo mistakes.
- All git commands begin with `git` followed immediately by an argument for the type of command you want to execute.
- For the right-hand-side of the diagram, the following are the useful git commands:

  - `git init`: Initialize a git repository. *Only do this once per project.*
  - `git status`: Show which files are staged in your working directory, and which are modified but not staged.
  - `git add`: Add modified files from your working directory to the stage.
  - `git diff`: Look at how files in the working directory have been modified.
  - `git diff --staged`: Look at how files in the stage have been modified.
  - `git commit -m "[descriptive message]"`: commit your staged content as a new commit snapshot.

Initialize a repository

- Git needs to be told that a folder is a repo. Otherwise, it won't keep files under version control.
- In this class, you won't need to tell git this (I'll tell git this), but in the real world you will. So we'll go over how to do this on GitHub and on the terminal.

On the terminal

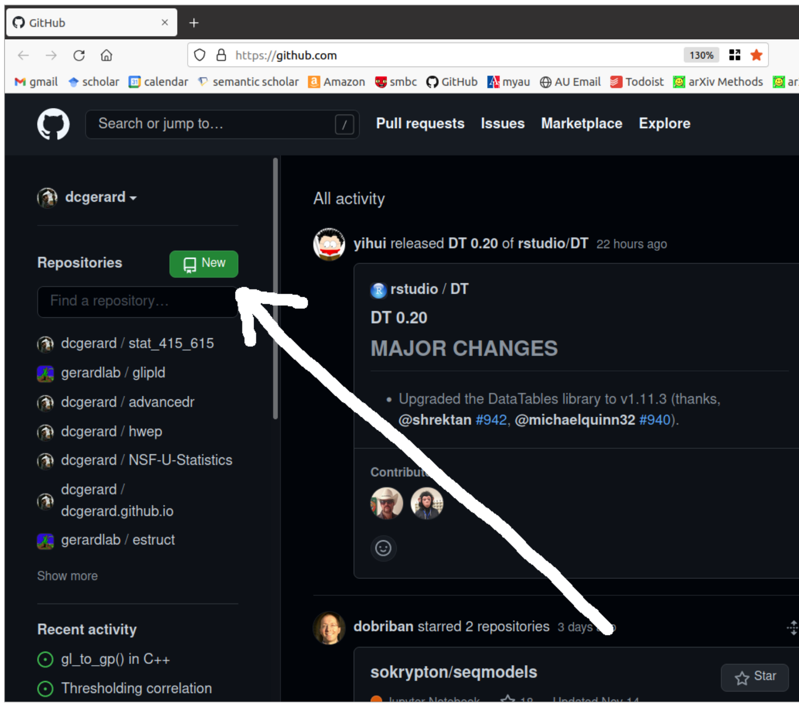
- Don't initialize on your local for this lecture. These are just the steps you would do if you needed to initialize on your local.
- Use `cd` to enter the folder that you would like to keep under version control.
- The use `git init`

```
git init
```

- This will tell git that the folder is a single repo.
- Your files are not yet tracked. You'll need to do the steps below to tell git which files to track. But at least git now knows that this is a repo where tracking is possible.

On GitHub

- Git is a version control system, GitHub is a website that hosts git repositories. (so on your resume, say that you know git, not GitHub).
- You can create a git repo on GitHub (GitHub's server is called the "remote"), then download ("clone") the repo onto your computer (your computer is called the "local").



dcgerard / stat_415_615
gerardlab / glipld
dcgerard / advancedr
dcgerard / hwep
dcgerard / NSF-U-Statistics
dcgerard / dcgerard.github.io
gerardlab / estruct

gl_to_gp() in C++
Thresholding correlation

yihui released DT 0.20 of rstudio/DT 22 hours ago

rstudio / DT
DT 0.20
MAJOR CHANGES
- Upgraded the DataTables library to v1.11.3 (thanks, @shrektan #942, @michaelquinn32 #940).

dobriban starred 2 repositories 3 days ago
sokrypton/seqmodels

- On your GitHub homepage, click on "New"
- Fill out the form. The options are pretty self-explanatory, and GitHub does a good job of providing descriptions. For this lecture, make sure
  - Repository name is "test".
  - The repo is set to be "Private"
  - You check "Add a README File"
- Click on "Create Repository.

Cloning

- "Cloning" is a fancy way to say download from GitHub.
- But it also means that your local copy is connected to the remote copy automatically.

- There are three ways you can clone a repository Enter the repo you want to clone, then click on the Button

  1. Using Download Zip

  * You just need to click on the Download Zip and you will get a zip file on your local machine.

  2. Git using "HTTPS"

  * Make sure that "HTTPS" is highlighted.
  * Then click on the button to copy the link.

  * Open up the terminal and type (**Make sure to change the link to what you copied**)

```
    git clone https://github.com/Semiyari/test.git
```

- As you have noticed that git command always starts with `git`. The `clone` command tells to download from github a copy to our local machine. You need to paste the URL to let git knows the location of the file on github.
- Then it will create a folder with the same name as whatever, your repository was called.
- Then change to the directory to

```
cd test
```

- If you type

```
ls -la
```

It will shows all files an folder on your repository - If you type

```
dir
```

It is similar to `ls` command but its most purpose is to show the directory of the contents.

- Let us view the `README.md` file

```
cat README.md
```

You may get something like

```
# test
```

There is only one line in my `README` file. If I go to my GitHub and open the README file on my repo I will get the samething. - Type

```
pwd
```

It will get me the directory that I am in on my terminal.

- Type

```
git log
```

You are going to get all sorts of logs. We can see commits - commit messages- the authors of the commits. You can get the etire history of your repo.

- **But you most likely will get the following error after running the code**'git clone https://github.com/Semiyari/test.git'.

```
    remote: Invalid username or password.
fatal: Authentication failed for 'https://github.com/Semiyari/test.git/'
```

why?
- The reason for this error is that you most likely haven't Generate a Personal Access Token (PAT). Read previous note to learn how to do it or you may want to watch a video that professor Dickens has provided

- Follow the Instructions in the video (whose link is provided below) in order to create your Personal Access Token. He also explained more about Git commands that I will sahre with you later.

    3. Git using "SSH"

    Professor Gerard has explained how to do it in below:

- Make sure that "SSH" is highlighted.
- Then click on the button to copy the link.
- In the terminal, navigate to where you want to download the repo, then clone it with `git clone`

```
    git clone git@github.com:dcgerard/test.git
```

```
Make sure to change the link to what you copied (don't use my link above).
```

- Then move into your new repo

```
ls
cd test
```

Status

- Use `git_status` to see what files git is tracking and which are untracked.

```
git status
```

- Git should tell you that everything is up-to-date

```
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

- Edit the README.md file to include your name, so that it looks something like this:

```
# test
David Gerard

Repo for trying out GitHub.
```

```
Make sure to save your changes.
```

- Now check the status again.

```
git status
```

- Git should be telling you that README.md has been modified, and the changes are not yet committed.

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
```

```
    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

- Add a new file, called "empty.txt" by

```
touch empty.txt
```

- **Exercise**: Check the status again. What do you notice?

Staging

- Use `git add` to add files to the stage.

```
git add README.md
```

- Always check which files have been added:

```
git status
```

- Useful flags for `git add`:
  - `--all` will stage all modified and untracked files.
  - `--update` will stage all modified files, but only if they are already being tracked.

Committing

- The git commit command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as "safe" versions of a project
  - Git will never change them unless you explicitly ask it to. Prior to the execution of git commit, The `git add` command is used to promote or 'stage' changes to the project that will be stored in a commit. These two commands `git commit` and `git add` are two of the most frequently used.
- Use `git commit` to commit files that are staged to the commit history.

```
git commit -m "Add name to README.md."
```

- Your message (written after the `-m` argument) should be concise, and describe what has been changed since the last commit.

- If you forget to add a message, git will open up your default text-editor where you can write down a message, save the file, and exit. The commit will occur after you exit the text editor.
- If your default text editor is vim, you can exit it using this.
- `git status` should no longer have README.md as a modified file.

```
git status
```

Common options

- `git commit`: Commit the staged snapshot. This will launch a text editor prompting you for a commit message. After you've entered a message, save the file and close the editor to create the actual commit.
- `git commit -a`: Commit a snapshot of all changes in the working directory. This only includes modifications to tracked files (those that have been added with git add at some point in their history).
- `git commit -am "'<Type your message>"`: A power user shortcut command that combines the -a and -m options. This combination immediately creates a commit of all the staged changes and takes an inline commit message.
- `git commit --amend`: This option adds another level of functionality to the commit command. Passing this option will modify the last commit. Instead of creating a new commit, staged changes will be added to the previous commit. This command will open up the system's configured text editor and prompt to change the previously specified commit message
- Click here

History of Changes

- You can use `git log` to see what commits you have done.

```
git log
```

- There should be only two commits right now. One from GitHub and one from adding the name to README.md.

```
commit 0301eeaf74062f0b80fdb3c27a60cc5ac6f28ca7 (HEAD -> main)
Author: dcgerard <gerard.1787@gmail.com>
Date:   Tue Nov 16 10:53:42 2021 -0500

    Add name to README.md

commit fefbaffe03e0b074c33aa215d1135e6f8b68701d (origin/main, origin/HEAD)
Author: David Gerard <gerard.1787@gmail.com>
Date:   Tue Nov 16 10:04:47 2021 -0500

    Initial commit
```

- **Exercise**: Add the following line of text to "empty.txt"

  ```
  blah blah blah
  ```

  Save the output. Now stage and commit the changes.

Looking at differences

- Add the following lines of text to README.md

  ```
  Never and never, my girl riding far and near
  In the land of the hearthstone tales, and spelled asleep,
  Fear or believe that the wolf in a sheep white hood
  Loping and bleating roughly and blithely leap,
  My dear, my dear,
  Out of a lair in the flocked leaves in the dew dipped year
  To eat your heart in the house in the rosy wood.
  ```

  And delete the line

  ```
  Repo for trying out GitHub.
  ```

- Use `git diff` to see changes in all modified files.

```
git diff
```

- Lines after a "+" are being added. Lines after a "-" are being removed.
- You can exit `git diff` by hitting `q`.
- `git diff` won't check for changes in the staged files by default. But you can see the differences using `git diff --staged`.

```
git diff
git diff --staged
```

- **Exercise**: Stage and commit your changes.

Pushing

- Use `git push` to push commits to GitHub.

```
git push origin main
```

```
Do this now.
```

- "origin" is the name of the remote.
- "main" is the name of the branch we are pushing to remote.
- You can see what the remote is named by typing

```
git remote -v
```

- You can see what branch you are on by

```
git branch
```

Pulling

- If a colleague has pushed changes to GitHub, you'll need to pull those changes ontol your local before you can push anything to GitHub.
- This is different than cloning. "Cloning" downloads a repo that wasn't on your local machine. "Pulling" updates your local machine with the changes on the remote.
- Use `git pull` to pull changes.

```
git pull origin main
```

- "origin" is the name of the remote.
- "main" is the name of the branch we are pulling to.
- If there are no changes on the remote, you'll get the following message

```
From github.com:dcgerard/test
 * branch            main       -> FETCH_HEAD
Already up to date.
```

Branching

- A branch is an "alternative universe" of your project, where you can experiment with new ideas (e.g. new data analyses, new data transformations, new statistical methods). After experimenting, you can then "merge" your changes back into the main branch.
- Branching isn't just for group collaborations, you can use branching to collaborate with yourself, e.g., if you have a new idea you want to play with but do not want to have that idea in main yet.
- The "main" branch (the default in GitHub) is your best draft. You should consider anything in "main" as the best thing you've got.
- The workflow using branches consists of

  1. Create a branch with an informative title describing its goal(s).
  2. Add commits to this new branch.
  3. Merge the commits to main

Create a branch

- You create a branch with the name `<branch>` by

```
git branch <branch>
```

- Suppose we wanted to calculate some summary statistics, but we are not sure if we want to include these in the report. Let's create a branch where we explore these summary statistics.

```
git branch sumstat
```

- You can see the list of branches (and the current branch) with

```
git branch
```

Move between branches

- You switch between branches with:

```
git checkout <branch>
```

- Move to the sumstat branch with

```
git checkout sumstat
```

Edit Branch

- When you are on a branch, you can edit and commit as usual.

Push branch to GitHub

- You can push your new branch to GitHub just like you can push your main branch to GitHub:

```
git push origin <branch>
```

Merge changes into main

- Suppose you are satisfied with your changes in your new branch, then you'll want to merge these into the main branch. You can do this on GitHub (see here). If you do so, then don't forget to pull the changes from main back into your local machine.

```
git pull origin main
```

- Alternatively, you can merge the changes in your local machine. First, checkout the main branch.

```
git checkout main
```

- Then use `merge` to merge the changes from `<branch>` into main.

```
git merge sumstat
```

- Don't forget to push your changes to GitHub

```
git push origin main
```

Resolving Merge Conflicts

- If two branches with incompatible histories try to merge, then git does not merge them.
- Instead, it creates a "merge conflict", which you need to resolve.
- Instructions on resolving merge conflicts can be found here.

List of git commands

- `git init`: Initialize a git repository. *Only do this once per project.*
- `git status`: Show which files are staged in your working directory, and which are modified but not staged.
- `git add`: Add modified files from your working directory to the stage.
- `git diff`: Look at how files in the working directory have been modified.
- `git diff --staged`: Look at how files in the stage have been modified.
- `git commit -m "[descriptive message]"`: commit your staged content as a new commit snapshot.
- `git clone <url>`: Download a fresh copy of a remote repository onto your local machine.
- `git remote add <remote> <url>`: Link a local repository with a remote repository. The nickname of the remote repository is `<remote>`.
- `git push <remote> <branch>`: Push the changes from branch `<branch>` to the remote repository named `<remote>`
- `git pull <remote> <branch>`: Pull any modifications from `<remote>` into your local machine in branch `<branch>`.
- `git branch <branch>`: Create a branch called `<branch>`. Note that the default branch is called `main` or `master`.
- `git checkout <branch>`: Move to the `<branch>` branch.
- `git merge <branch>`: Merge the changes in `<branch>` into the current branch.

Vocabulary List (Blischak et. al., 2016)

- **Version Control System** (VCS): (*noun*) a program that tracks changes to specified files over time and maintains a library of all past versions of those files
- **Git**: (*noun*) a version control system

- **repository (repo)**: (*noun*) folder containing all tracked files as well as the version control history
- **commit**: (*noun*) a snapshot of changes made to the staged file(s); (*verb*) to save a snapshot of changes made to the staged file(s)
- **stage**: (*noun*) the staging area holds the files to be included in the next commit; (*verb*) to mark a file to be included in the next commit
- **track**: (*noun*) a tracked file is one that is recognized by the Git repository
- **branch**: (*noun*) a parallel version of the files in a repository
- **local**: (*noun*) the version of your repository that is stored on your personal computer
- **remote**: (*noun*) the version of your repository that is stored on a remote server; for instance, on GitHub
- **clone**: (*verb*) to create a local copy of a remote repository on your personal computer
- **fork**: (*noun*) a copy of another user's repository on GitHub; (*verb*) to copy a repository; for instance, from one user's GitHub account to your own
- **merge**: (*verb*) to update files by incorporating the changes introduced in new commits
- **pull**: (*verb*) to retrieve commits from a remote repository and merge them into a local repository
- **push**: (*verb*) to send commits from a local repository to a remote repository
- **pull request**: (*noun*) a message sent by one GitHub user to merge the commits in their remote repository into another user's remote repository