Intro to Git

Welcome to the pit.

A presentation heavily borrowed from datacamp and cobbled together with some other things

Git Agenda

- Git Background
- Git Setup
- Git Workflow
- Git Collaboration

I’ve never heard of git

What even is git?

Twas the Night

Twas the Night V1

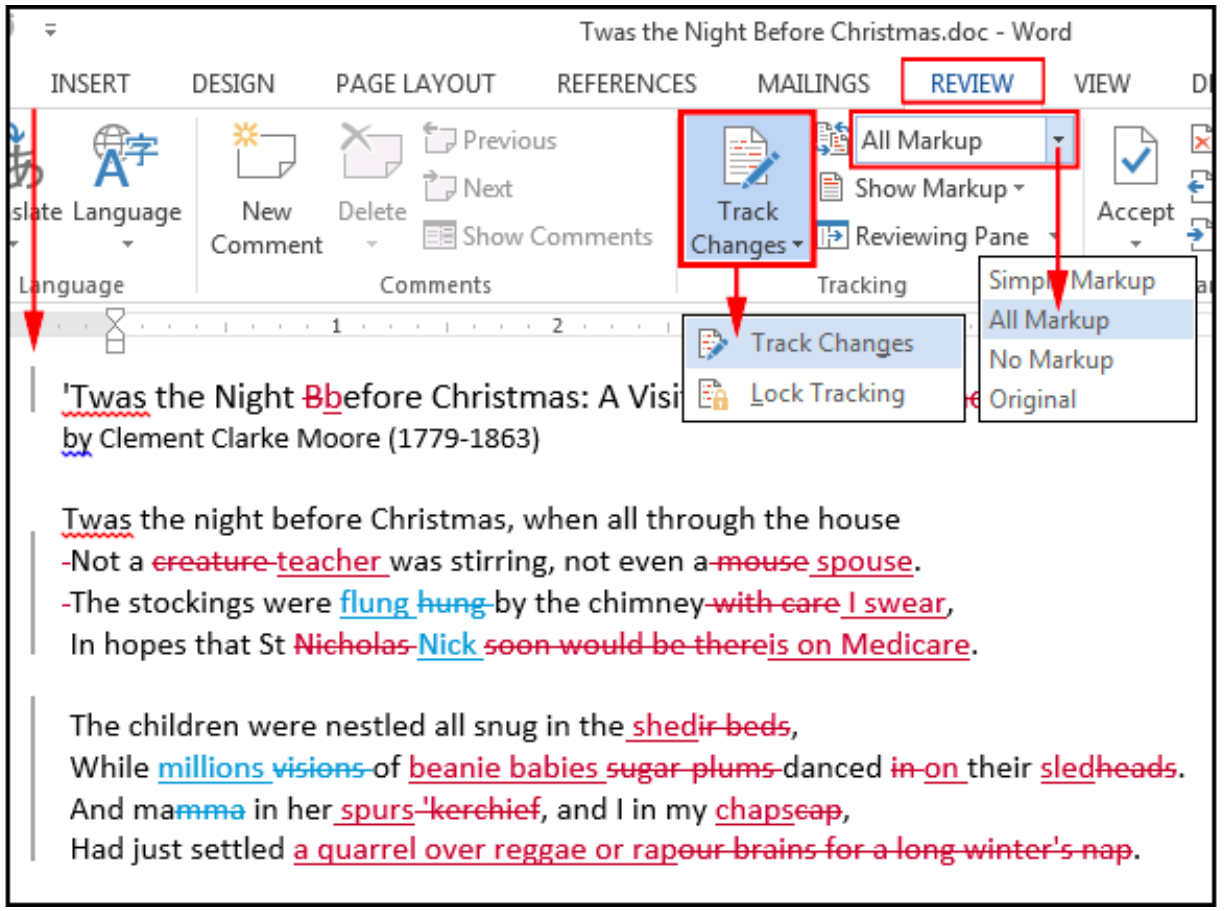
Twas the Night V2_DN

Twas the Night V2_DNSSedits

Twas the Night V2

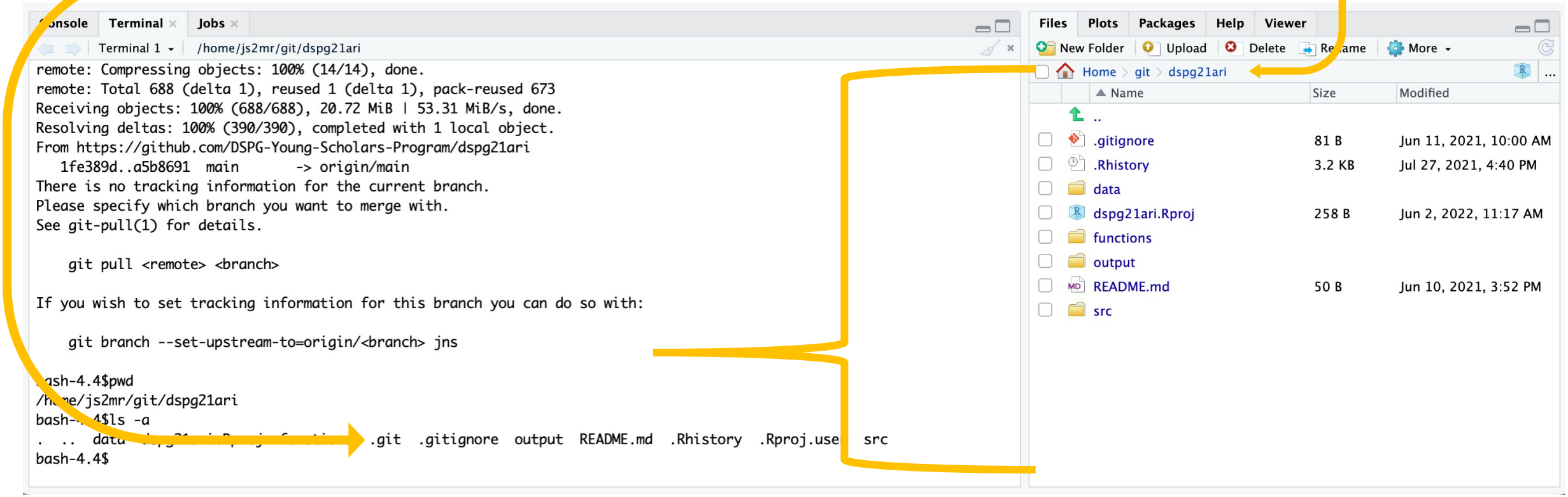
Twas the Night V3_DN

Twas the Night V1_2

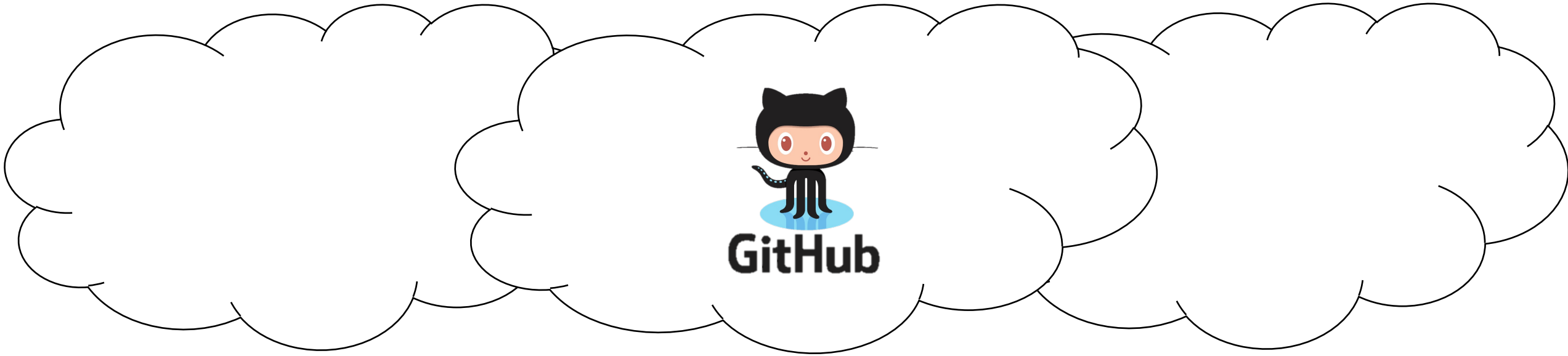


How do git repositories work?

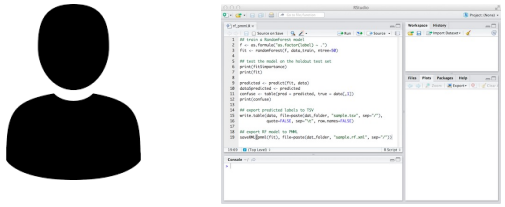
- Git repositories are basically project folders containing:
  - project directory: directory containing all the files and folders associated with your project
  - git directory: directory containing all the extra information required to capture changes between versions.



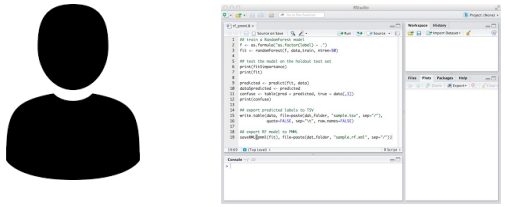
What even is git?



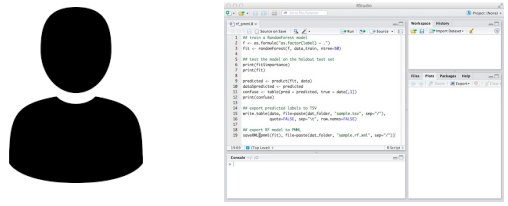
Gizem – Version 1



Devika – Version 2



Teja – Version 3



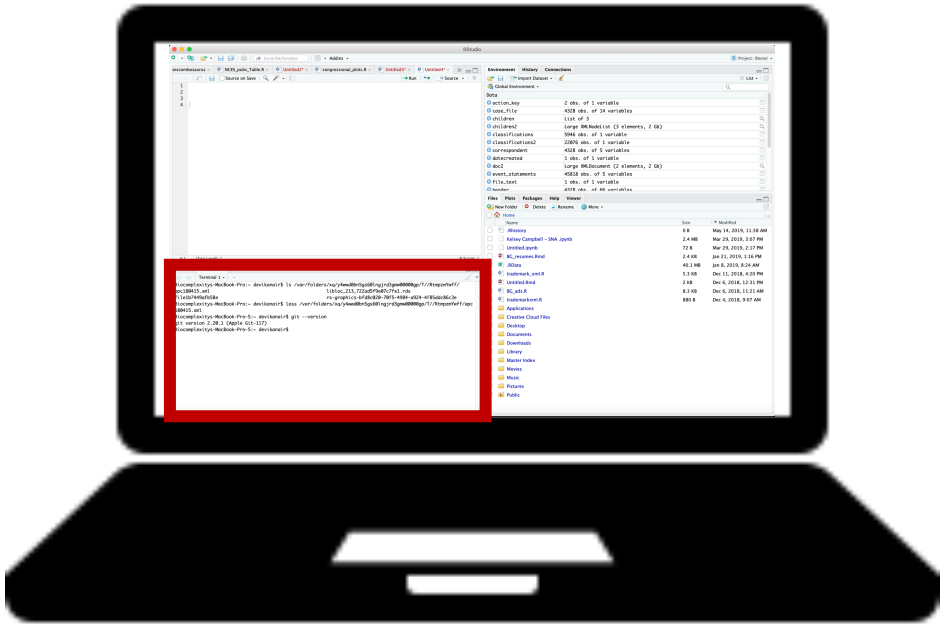
What even is git?

lib/compose.js — View file @e71062d

```
@@ -12,19 +12,27 @@ function setWritability(obj, writable) {
  });
}

origin//their changes                                   Use this
function mixin(base, mixins) {
  base.mixedIn = base.hasOwnProperty('mixedIn') ? base.mixedIn :
[];



  for (var i = 0; i < mixins.length; i++) {
    if (base.mixedIn.indexOf(mixins[i]) == -1) {
module.exports = {
  mixin: mixin
};
```

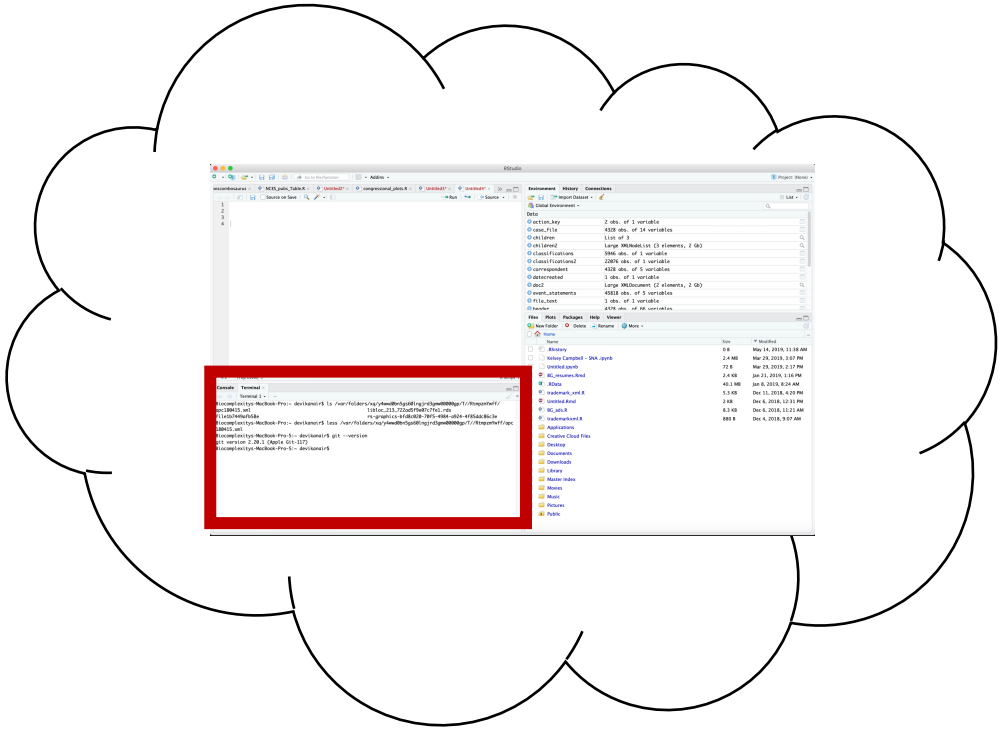
```
@@ -12,19 +12,27 @@ function setWritability(obj, writable) {
  });
}

HEAD//our changes                                       Use this
    function mixin(base, mixins) {
      base.mixedIn = Object.prototype.hasOwnProperty.call(base, 'mi
xedIn') ? base.mixedIn : [];

      for (var i = 0; i < mixins.length; i++) {
        if (base.mixedIn.indexOf(mixins[i]) == -1) {
          setWritability(base, false);
          mixins[i].call(base);
          base.mixedIn.push(mixins[i]);
        }
      }

  for (var i = 0; i < mixins.length; i++) {
    if (base.mixedIn.indexOf(mixins[i]) == -1) {
module.exports = {
  mixin: mixin
};
```

Where are we doing git?



Your personal laptop and RStudio application is for practice, personal projects.

All project-related code must be written to the RStudio server.

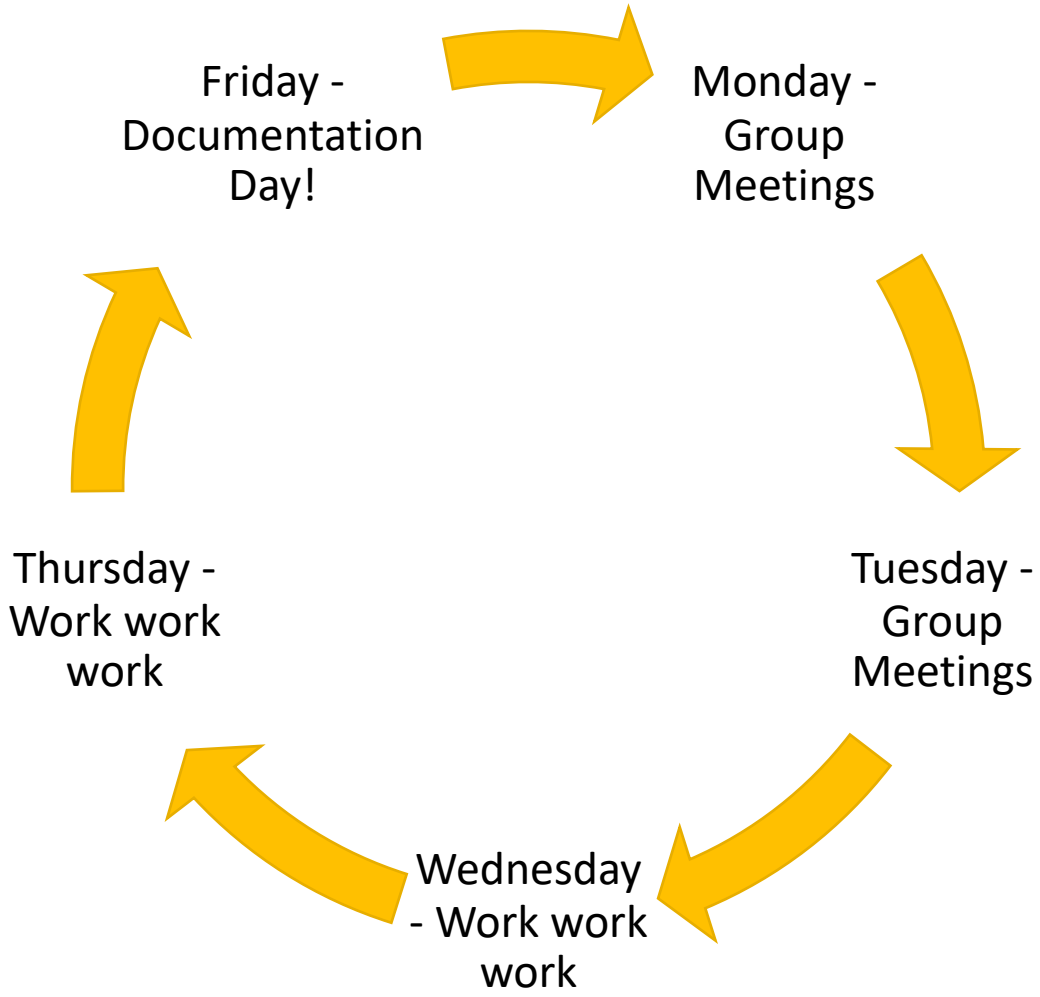


We will always do git in the terminal.

Your Role

Fellows responsible for reviewing and merging code?

- Does it run?
- No datasets are checked in?
- Are there loops that can be written more simply with an apply or map function?
- Is there repetitive code that could be written more simply using a custom function?
- Are your assumptions about the data tested in code?



Let’s Git It Started

Git Setup

Git Installation

Git Configuration

Git Repositories

Git On Up

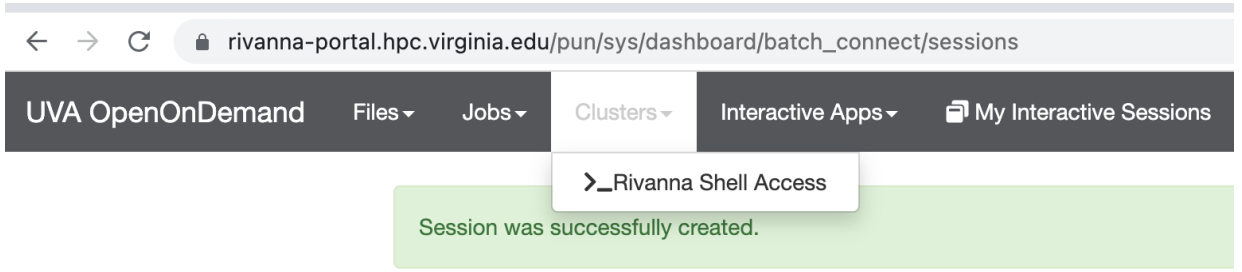
Git Installation

- Step 1: Check to see if you already have git installed.

  Open up a terminal window

  Enter the following

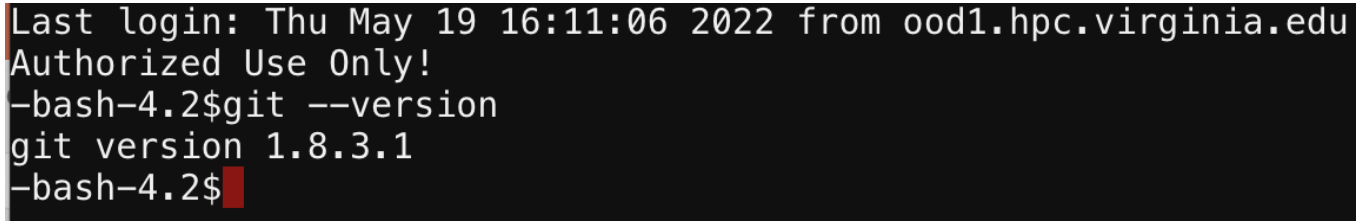
  ```
  git --version
  ```



- Results: If you see this, then:

  ```
  git version 1.8.3.1
  ```

  ✓ **YOU'RE GOOD**




```
'git' is not recognized as an internal or
external command, operable program or batch file
```

x Do not pass go, instead go here:

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

(I Can’t Git No) Satisfaction

Git Configuration

- Step 2: If you don’t already have a GitHub account, create one. If you do, log into it.

- Step 3: Check git setup on your computer

```
git config --list
```

- Step 4: Setup git on your computer

  Add your name and email to your settings **using your GitHub email account**

```
git config --global user.name “Cesar Montalvo”
git config --global user.email cpm9w@virginia.edu
```

Double check this change went through by using:

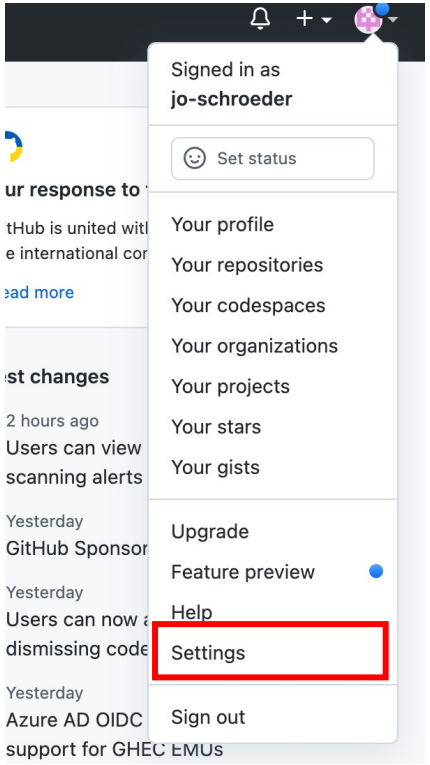
```
git config --global user.name
git config --global user.email
```

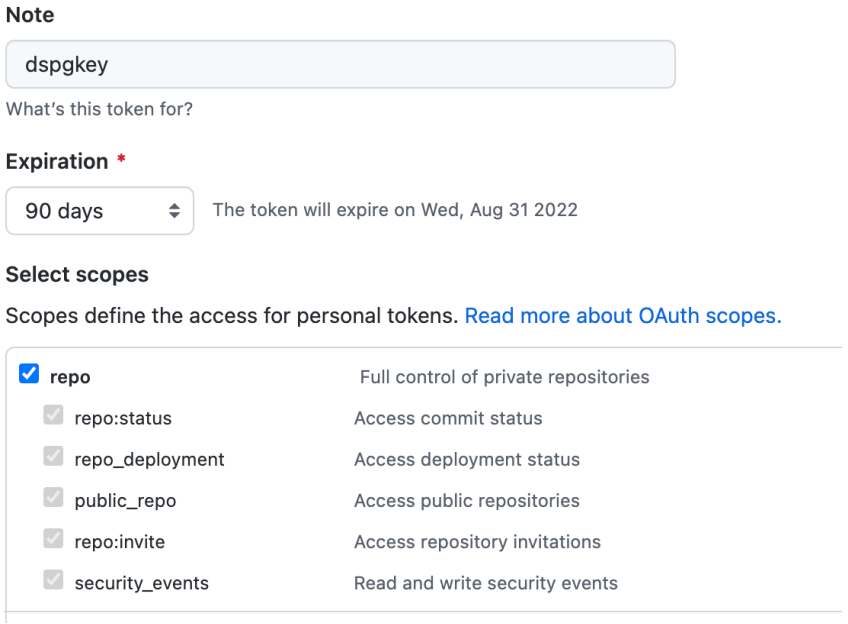
- Step 5: Navigate on GitHub:
  - User icon in top right -> Settings -> Developer settings -> Personal access tokens

- Step 6: Generate new token
  - Note: dspgkey
  - Expiration: 90 days
  - Scope: repo

  Generate token

- Step 7: Copy & save the key
  - Password manager
  - Environmental variable





How does git work?

Git Workflow

1. Write your code

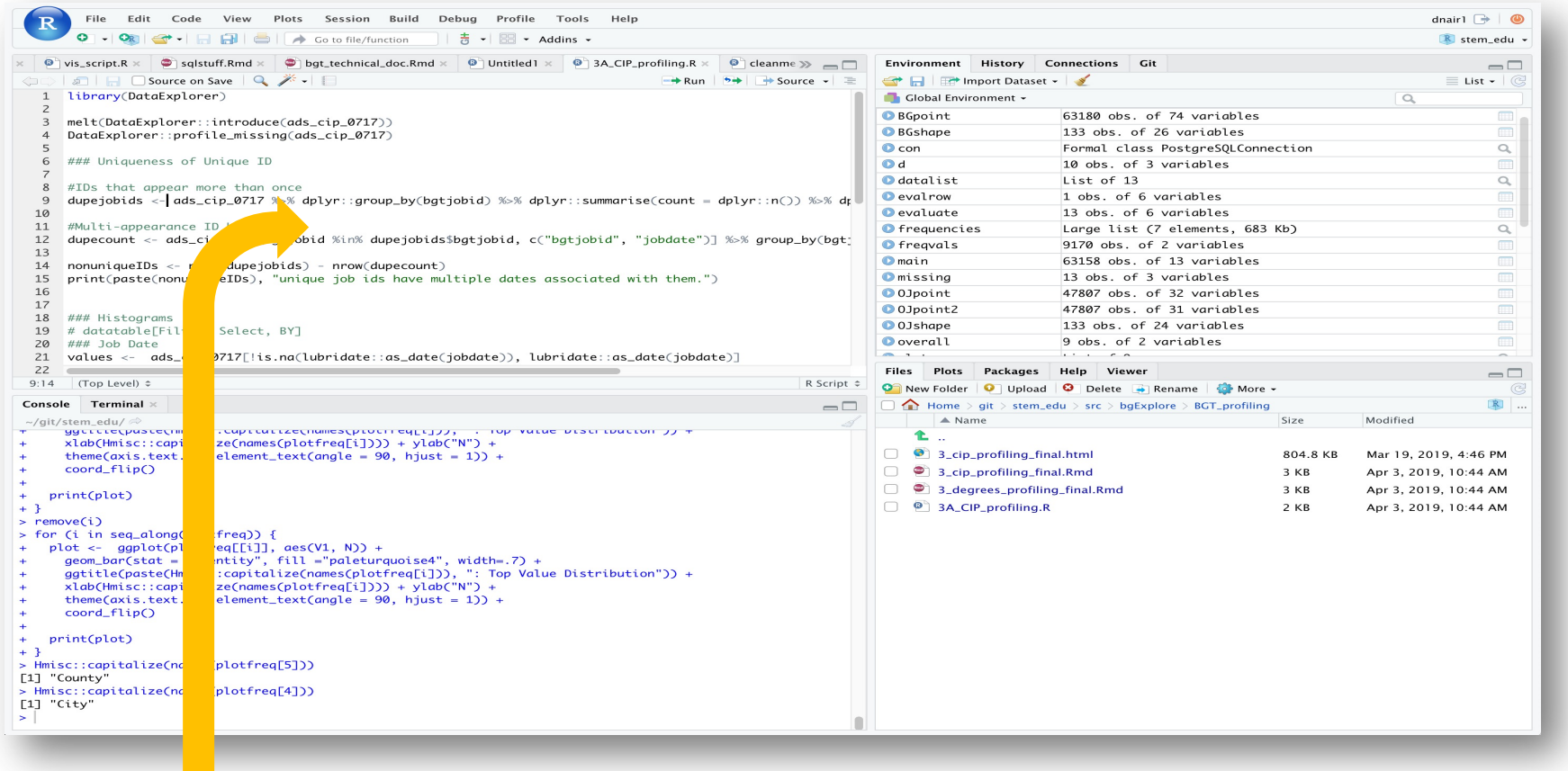
2. Stage your code

3. Commit your code

4. Push your code

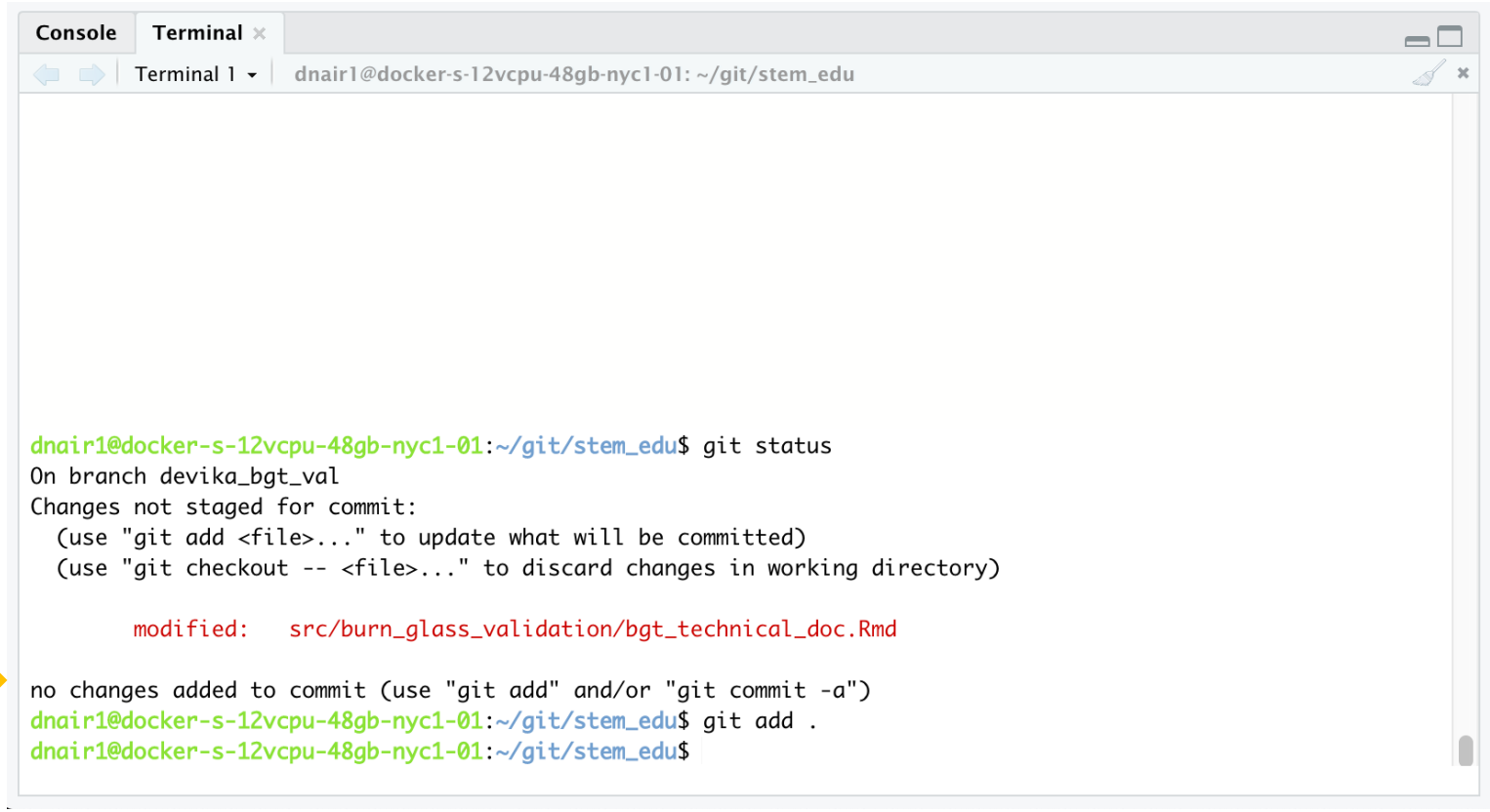
5. Merge your code

How does git work?



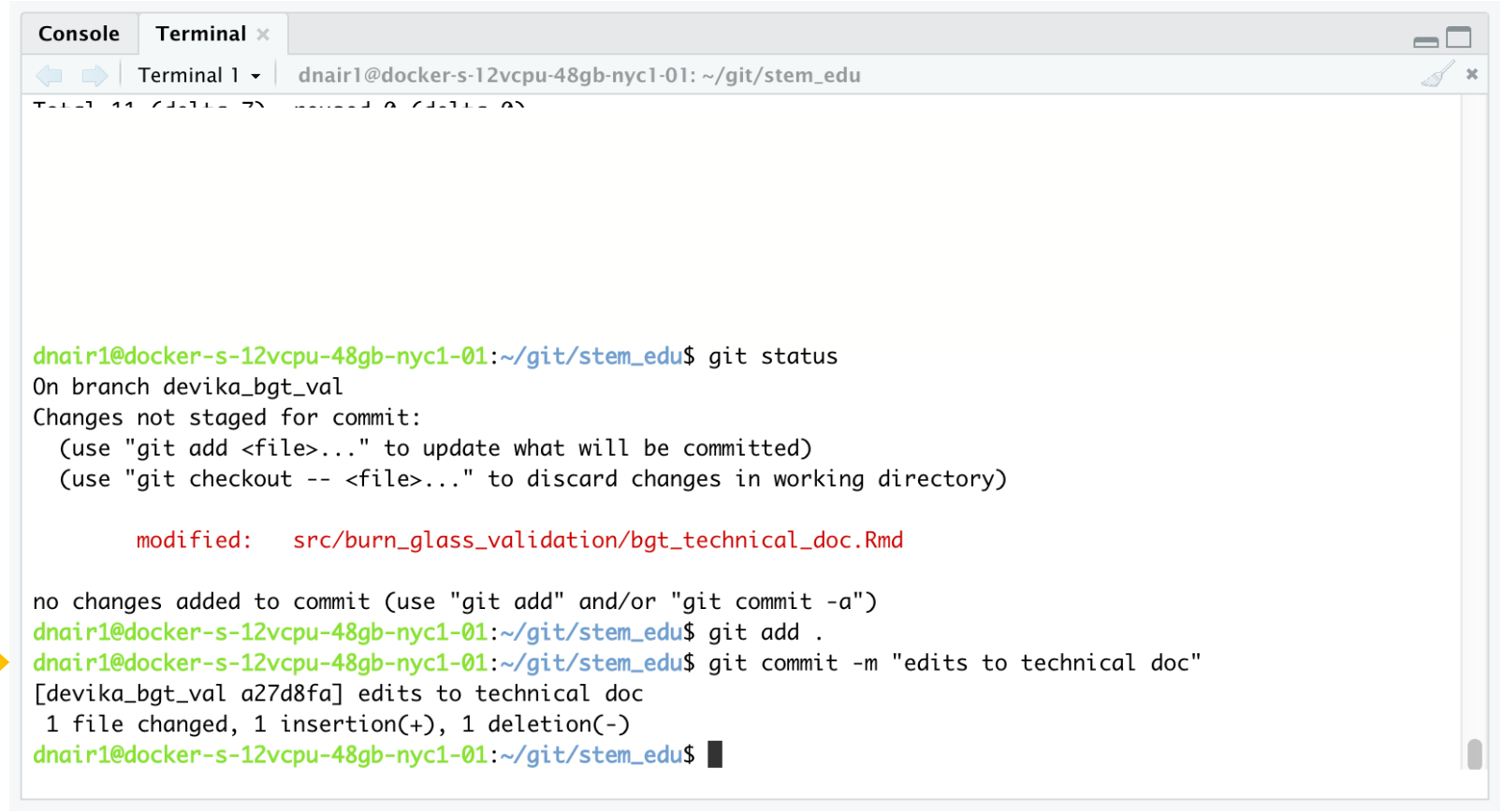
1. Write your code

How does git work?



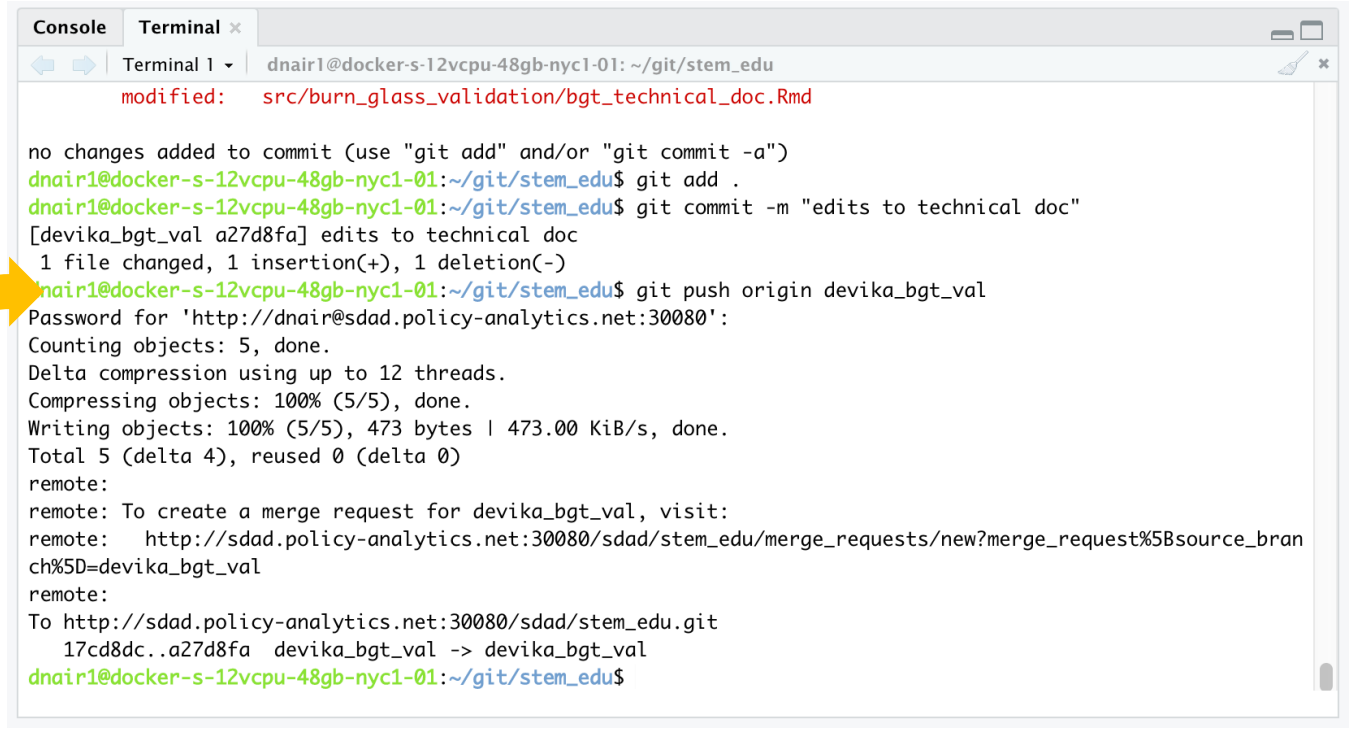
2. Stage your code

How does git work?



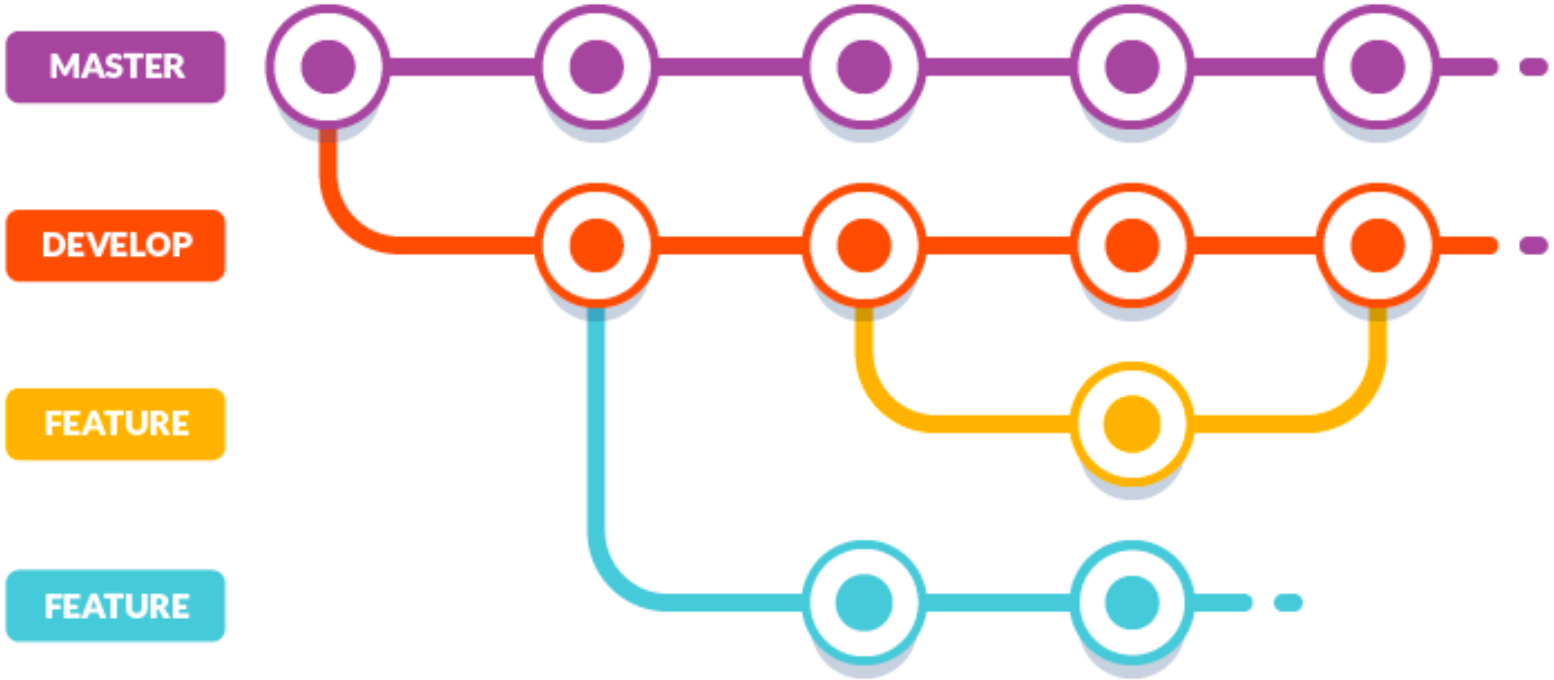
2. Commit your code

How does git work?



4. Push your code

Git Branches



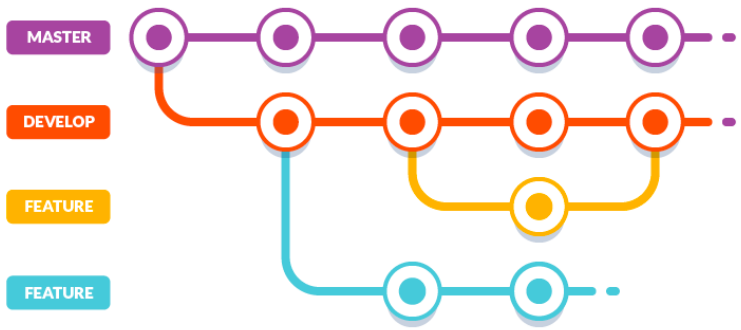
Master Branch:
Functional combination of all team members' code

Branch:
A copy of the code that is undergoing development

Branch:
A copy of the code that is undergoing development

Branch:
A copy of the code that is undergoing development

Git Branches



Master Branch:
Functional combination of all team members' code

Example:
Business Innovation – detecting innovation in text sources

Branch:
A copy of the code that is undergoing development

Gizem Task:
Gizem working to import journal articles text data

Branch:
A copy of the code that is undergoing development

Devika Task:
Devika working to articulate data structure and data types of articles

Branch:
A copy of the code that is undergoing development

Teja Task:
Teja working to import and clean FDA data for subsequent validation

Git Branches

Example:
Business Innovation – detecting innovation in text sources

Gizem Task: Gizem working to import journal articles text data 	**Branch Name:** factiva_gizem	**Scripts:** import_factiva.R import_factiva_meta.R
Devika Task: Devika working to articulate data structure and data types of articles 	**Branch Name:** factiva_devika	**Scripts:** import_factiva.R clean_factiva.R
Teja Task: Teja working to import and clean FDA data for subsequent validation 	**Branch Name:** fda_teja	**Scripts:** import_FDA.R clean_FDA.R

Save your work!
Branches are pushed up to GitHub for code review.

(git status, git add, git commit, git push)

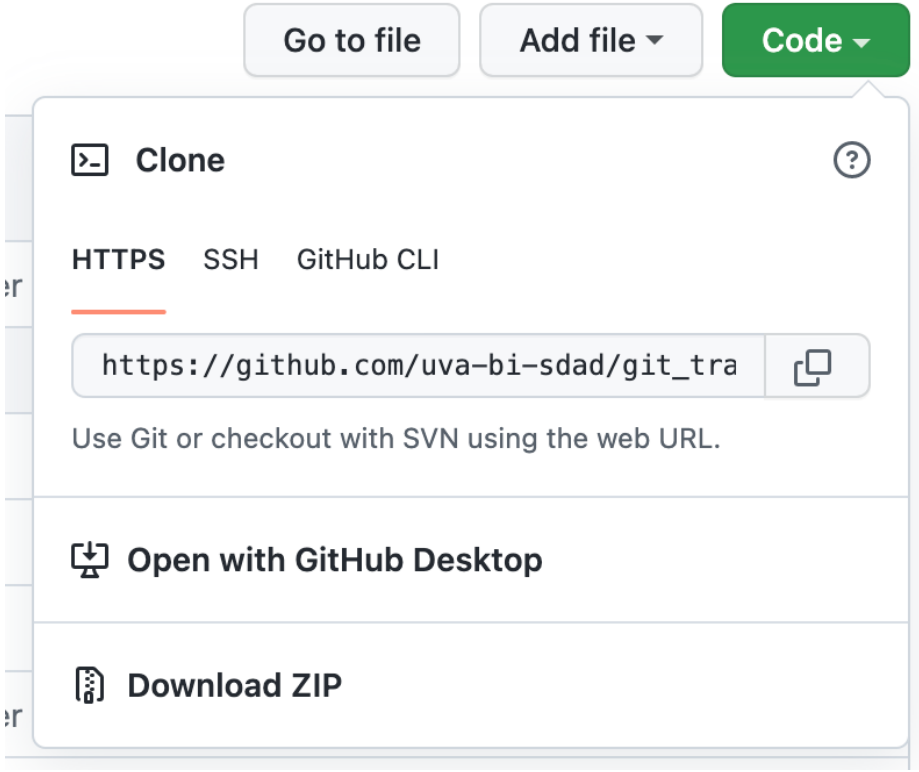
Exercise

Git Test Code

- Step 1: Find the git_training repository on GitHub (uva-bi-sdad organization)
  - Copy the SSH link

    `https://github.com/uva-bi-sdad/git_training.git`
- Step 2: Clone the project into your Rivanna RStudio container using Rstudio window controls
  - In your Home directory – create a new folder called ‘git’
  - File
  - New project
  - Version Control
  - Git

    Repository URL <- https://github.com/uva-bi-sdad/git_training.git
  - Create project as a subdirectory of your new ‘git’ folder within your home directory



Exercise

Git Test Code

- Step 3: What branch are you on?
  - In terminal Run git status
  - Create your branch git checkout –b *your_branch*
- Step 4: Create a new file in /dspg2022 and save it as *your_name.*txt
- Step 5: In terminal
  - Run git status <- What has been changed?
  - Run git add . <- Stage new changes
  - Run git commit –m “*your message*” <- commit your changes
  - Run git push origin *your_branch*

Exercise

Git Test Code

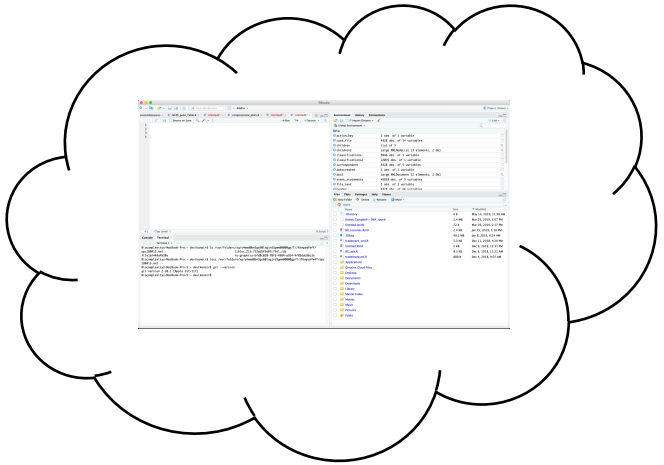
- Step 6: Navigate to GitHub
  - Open a pull request
- Step 7: Delete your branch

  PAUSE
- Step 8: Git pull everyone’s changes

Exercise – What just happened?

Git Test Code

You made changes to your code on the server



You sent those changes to GitHub via Terminal

GitHub now shows the latest version

This works when you are the sole writer of code on your project.

But what if you aren't?

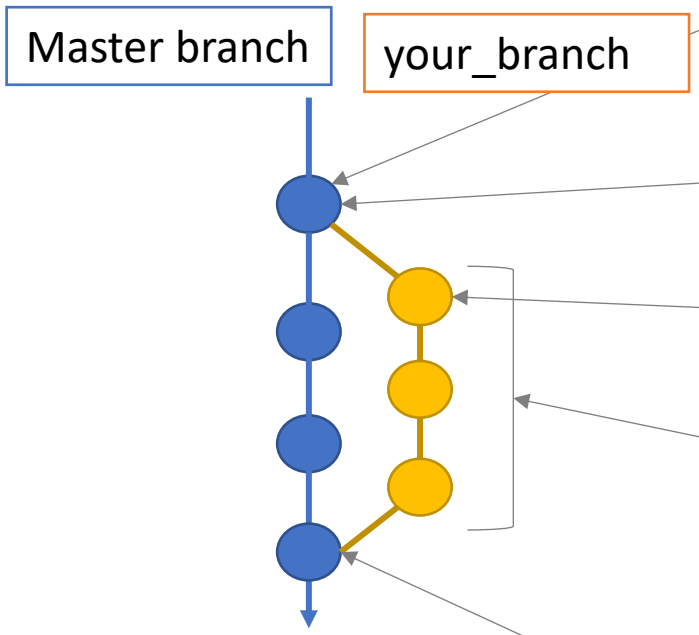
Just remember – BWPM

Branch → **Write** → **Push** → **Merge**

- **Branch**
  - Pull latest master
  - Create git branch
- **Write**
  - Write your code!
- **Push**
  - Add your changes
  - Commit the changes
  - Push your branch
- **Merge**
  - Review your code
  - Merge your branches
  - Merge branch into master
  - Delete your branch

Bran Writes Pushy Memos

GIT CHEAT SHEET



0. Make sure you are in your project repository, either of these two wavs:
- Navigate to your project repo by opening the .Rproj file in Files dspg21ari.Rproj
- Use Terminal command line to:
  - To check your working directory: `pwd`
  - To see contents of folder: `ls -a`
  - To move into folder below: `cd foldername`
  - To move into folder above: `cd ..`

1. Make sure your master branch is up to date
   1. Go to the master branch: `git checkout master`
   2. Update your master branch: `git pull origin master`

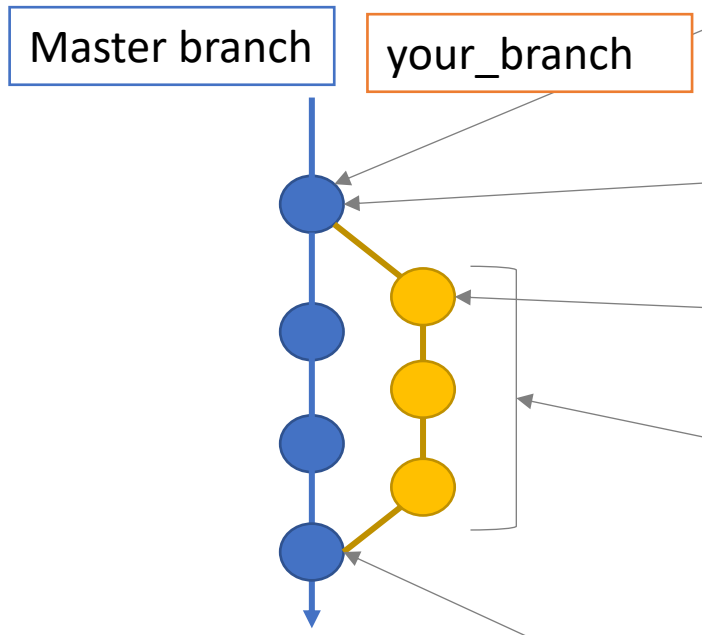
2. Create a new branch: `git checkout -b your_branch`
   1. Add your name or initials as the branch name

dspg21ari.Rproj

3. Go code and write commits!
   1. `git add <my file>`
   2. `git commit -m 'this is what I did!'`
4. Push your branch `git push origin your_branch`

5. Issue a pull request
6. Have someone (maintainers) review your code
7. (Maintainers) merge the pull request
   1. You can also checkoff a box that will also delete the branch on the remote
   2. Or delete the branch manually under the branches view
8. Go back to master: `git checkout master`
9. Pull down your merged code: `git pull origin master`
10. Delete your branch: `git branch -d your_branch` (note it is a **lower case** d)
11. Clean up your branches: `git fetch --prune`

GIT CHEAT SHEET



0. Make sure you are in your project repository, either of these two wavs:
- Navigate to your project repo by opening the .Rproj file in Files dspg21ari.Rproj
- Use Terminal command line to:
  - To check your working directory: `pwd`
  - To see contents of folder: `ls -a`
  - To move into folder below: `cd foldername`
  - To move into folder above: `cd ..`

1. Make sure your master branch is up to date
   1. Go to the master branch: `git checkout master`
   2. Update your master branch: `git pull origin master`

2. Create a new branch: `git checkout -b your_branch`
   1. Add your name or initials as the branch name

3. Go code and write commits!
   1. `git add <my file>`
   2. `git commit -m 'this is what I did!'`
4. Push your branch `git push origin your_branch`

5. Issue a pull request
6. Have someone (maintainers) review your code
7. (Maintainers) merge the pull request
   1. You can also checkoff a box that will also delete the branch on the remote
   2. Or delete the branch manually under the branches view
8. Go back to master: `git checkout master`
9. Pull down your merged code: `git pull origin master`
10. Delete your branch: `git branch -d your_branch` (note it is a **lower case** d)
11. Clean up your branches: `git fetch --prune`

Git Lucky

Git Collaboration

Git Branches

Git Push

Git Merge

~~Git Pull~~

Git Branch Workflow

Git Branching

Check git branch

Create git branch

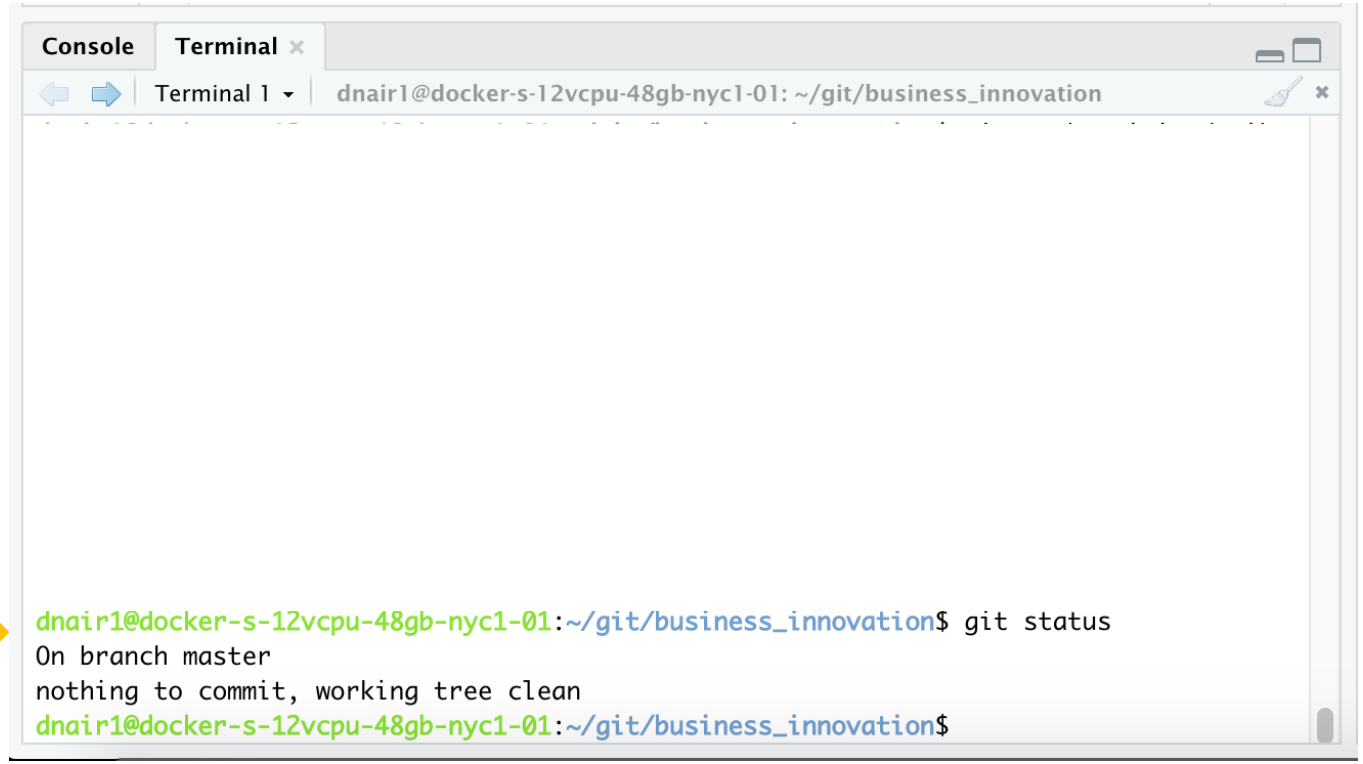
Write your code

Push your code

Merge your code

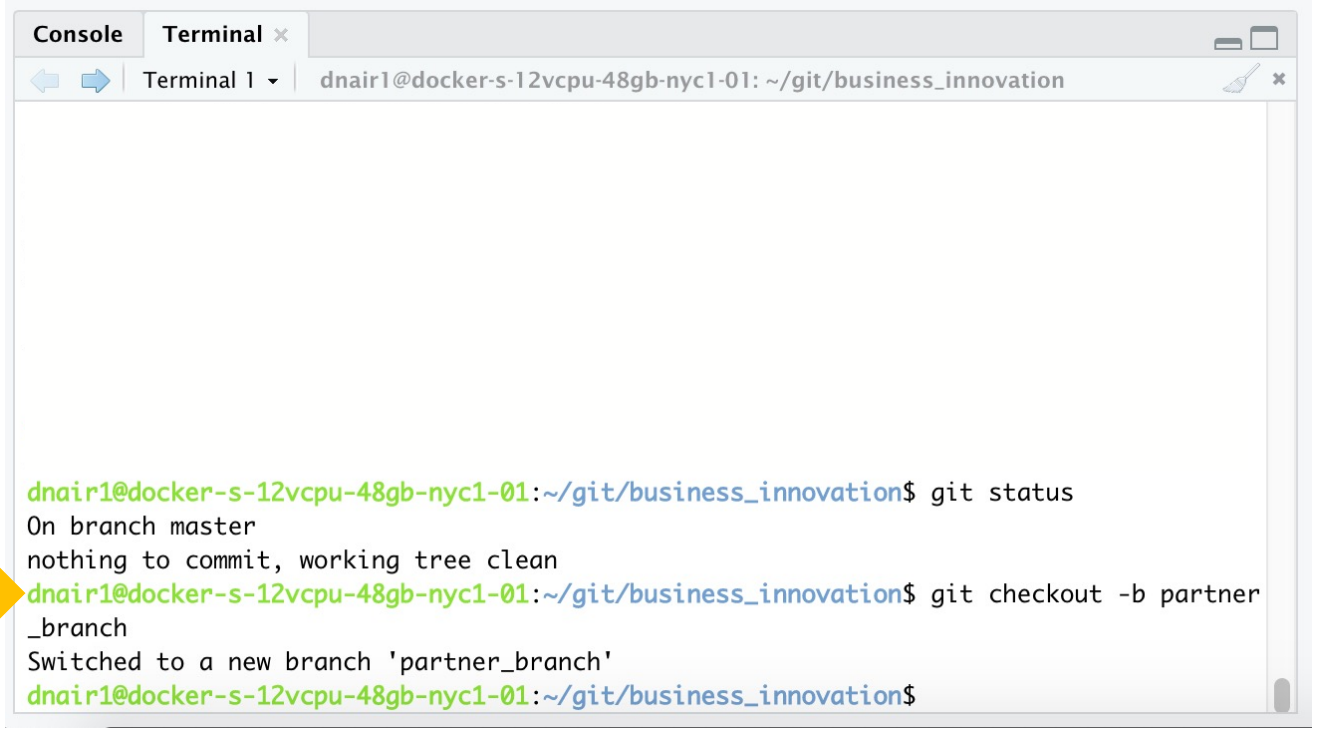
Note: you already know how to write and push your code!

How does git work?



1. Check git branch

How does git work?

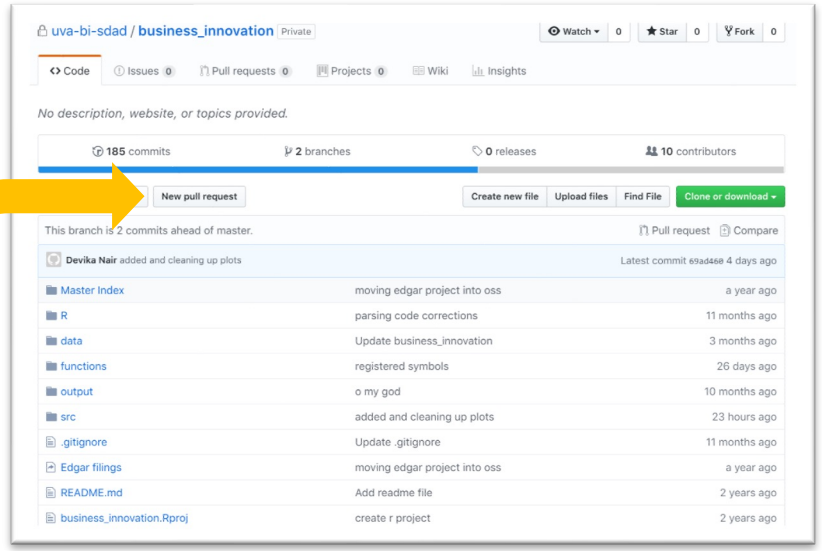


2. Create git branch

How does git work?

3. Write your code

4. Push your code



5. Merge your code

Exercise

Git Test Code

Check git branch

Create git branch

Write your code

Push your code

- Teams of 2
- Clone your partner's personal project (see Come Togither slide)
- Make partner collaborator on your project
- In Terminal
  - Check what branch you're on using git status
  - Create a new branch for yourself using git checkout -b partner_branch
- In Console
  - Open your partner's test.R script
  - Make a change to the script, like changing `mtcars` to `head(mtcars)`
- In Terminal
  - Check that the test.R script appears as modified using git status
  - git add .
  - git commit –m "updated partner's test script"
  - git push origin branchname

  Enter GitHub password
  - git status
- Navigate to GitHub – Is your newest branch available?

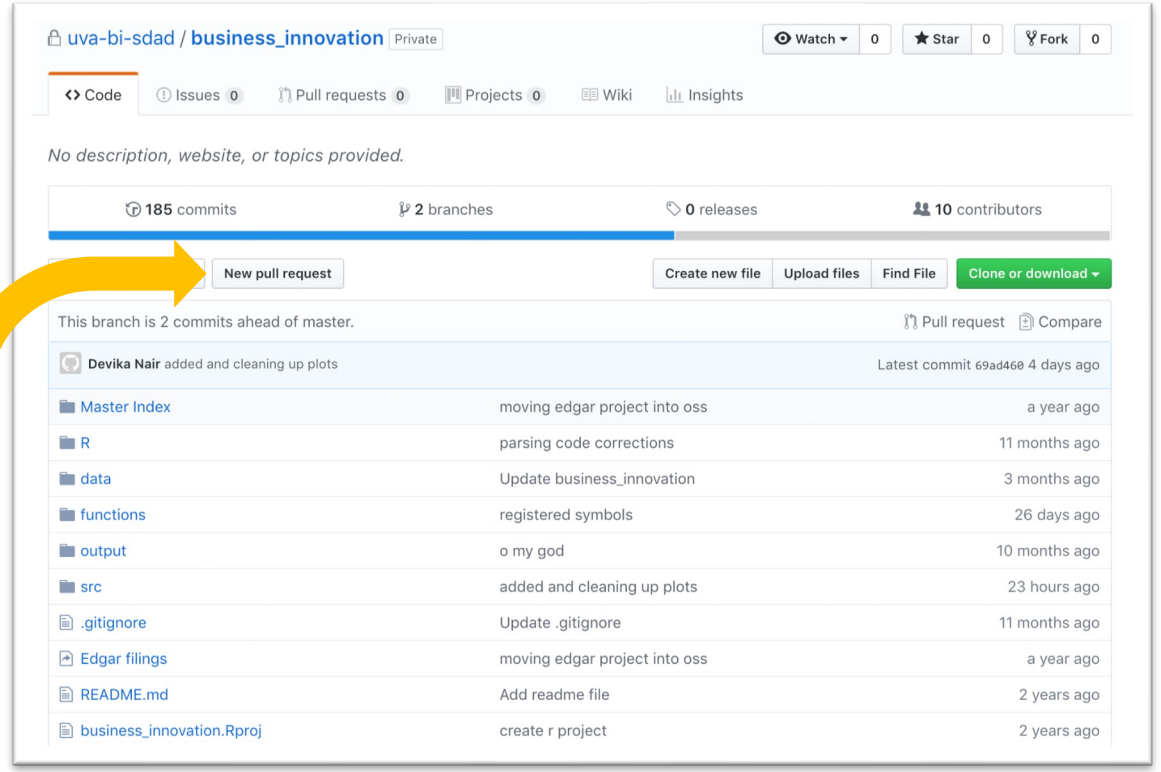
When your coworker asks you which git branch you're currently working on

Exercise - what did we just do?

Example:
Personal project

Partner A Task: Partner A looking at mtcars data 	**Branch Name:** master	**Scripts:** test.R	**Save your work!** Branches are pushed up to GitHub for code review. (git status, git add, git commit, git push)
Partner B Task: Partner B decided we only needed to look at first 6 rows of mtcars 	**Branch Name:** partner_branch	**Scripts:** test.R	

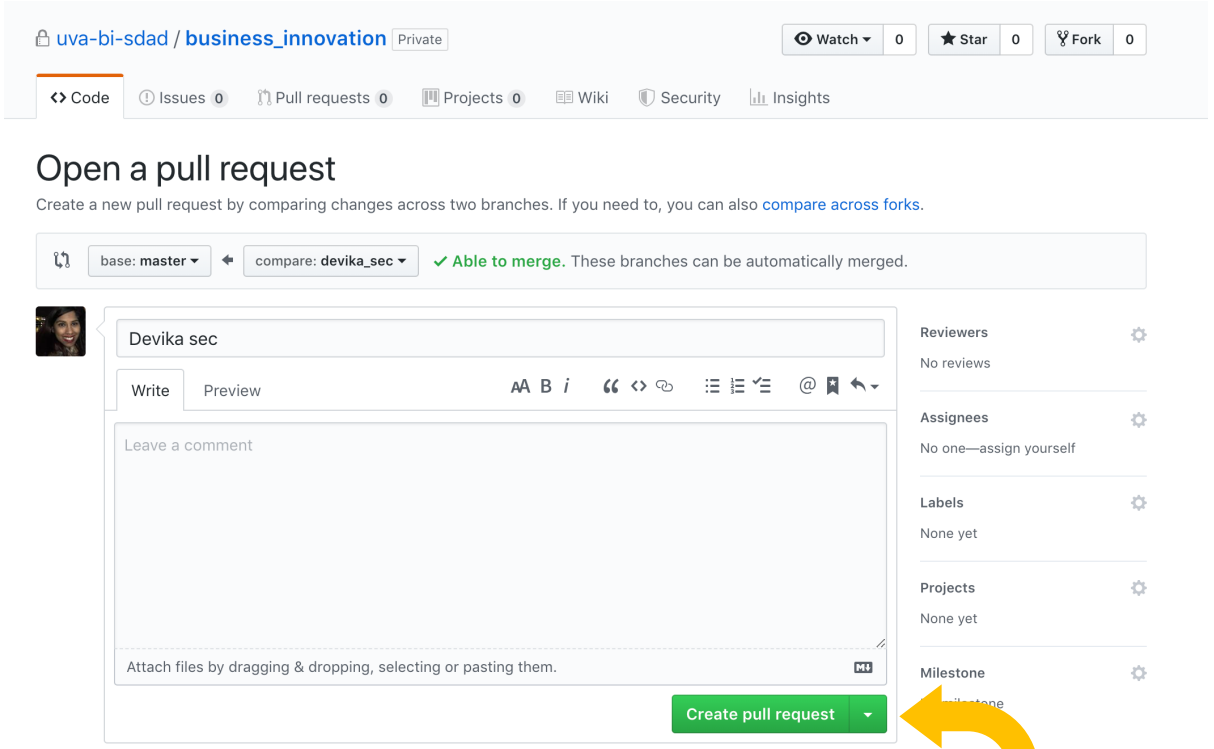
Git Merge



Create pull request

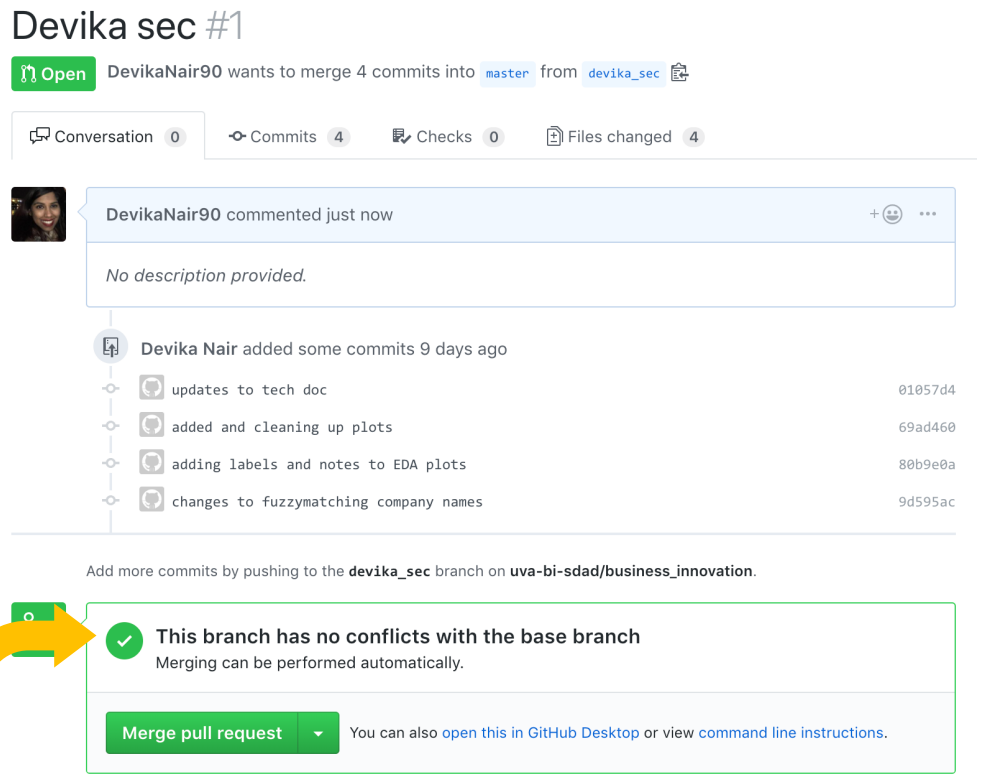
(pull request = merge request)

Git Merge



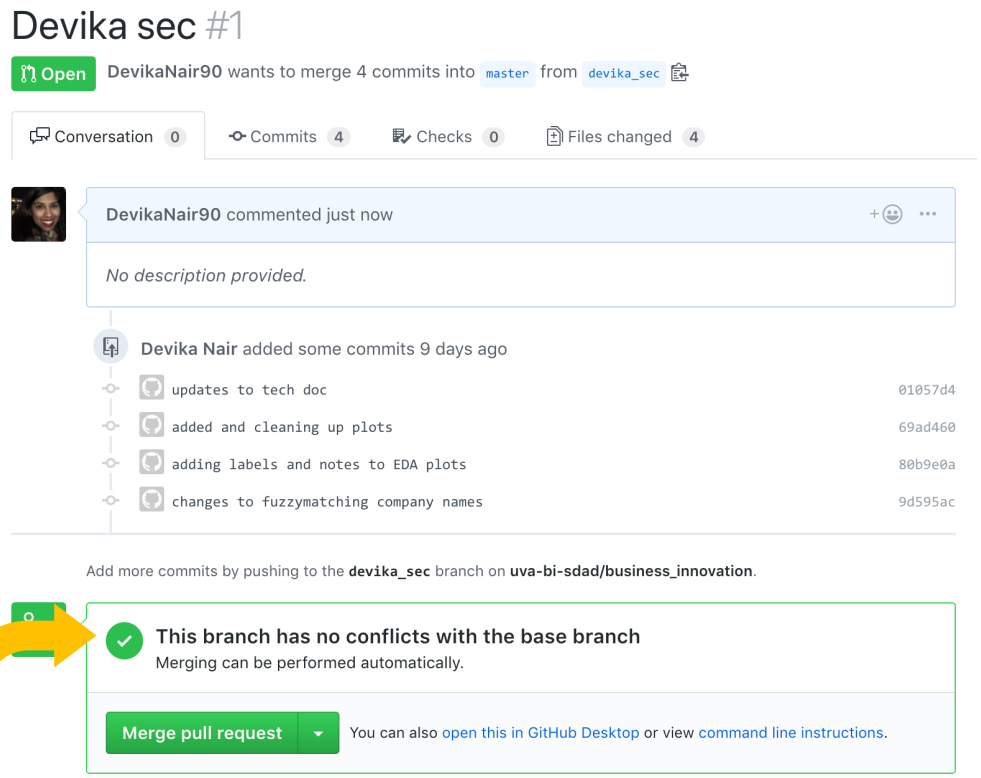
Create pull request

Git Merge



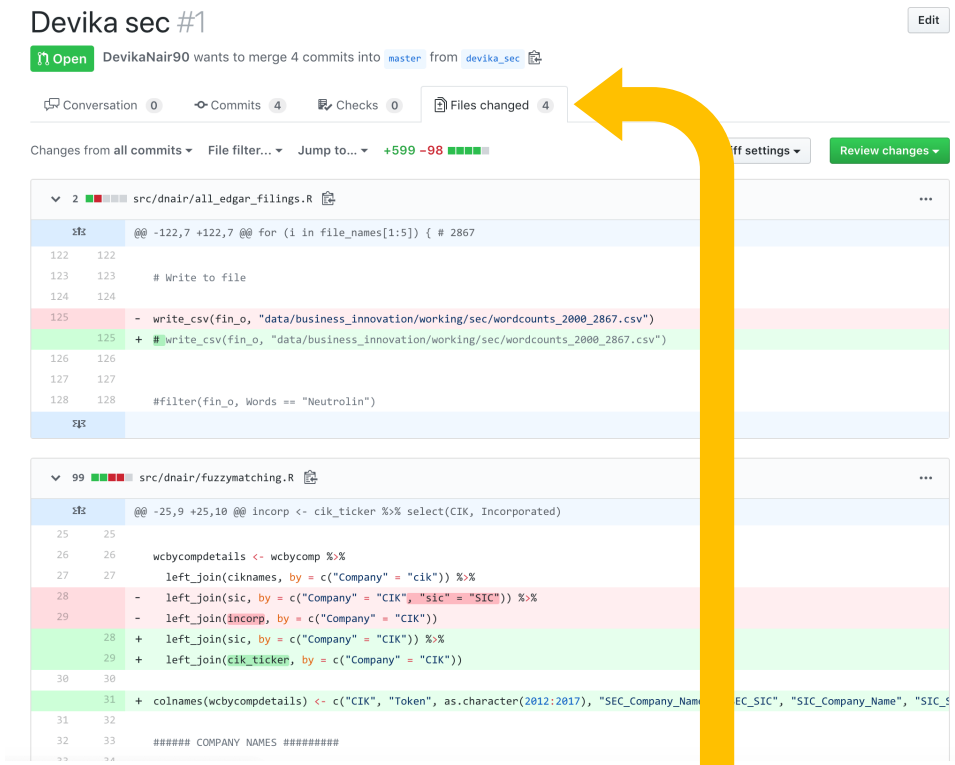
Check for conflicts

Git Merge



Check for conflicts

Git Merge



Review code changes

Just remember – BWPM

Branch → **Write** → **Push** → **Merge**

- **Branch**
  - Check git branch
  - Create git branch
- **Write**
  - Write your code!
- **Push**
  - Add your changes
  - Commit the changes
  - Push your branch
- **Merge**
  - Review your code
  - Merge your branches
  - Merge branch into master

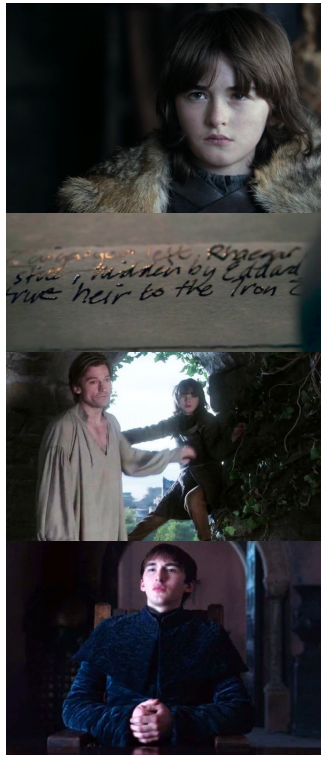
Bran Writes Pushy Memos

COVER YOUR EARS

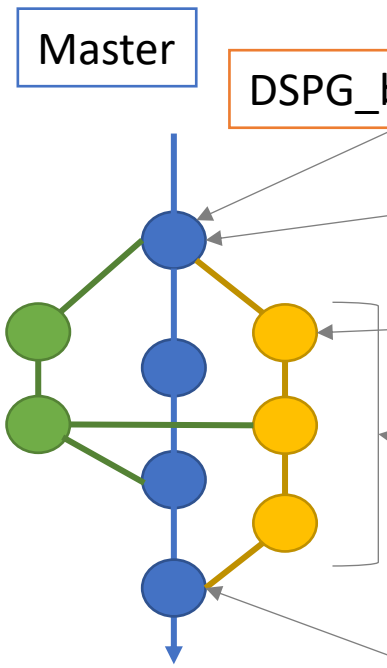
Just remember – BWPM

Branch → **Write** → **Push** → **Merge**

- **Branch**
  - Check git branch
  - Create git branch
- **Write**
  - Write your code!
- **Push**
  - Add your changes
  - Commit the changes
  - Push your branch
- **Merge**
  - Review your code
  - Merge your branches
  - Merge branch into master



Bran
Writes 0 ravens?
gets **Pushed**
Emerges winner



0. Make sure you are in your project repository, either of these two ways:
   - Navigate to your project repo by opening the .Rproj file in Files
   - Use Terminal command line to:
     - To check your working directory: pwd
     - To see contents of folder: ls -a
     - To move into folder below: cd foldername
     - To move into folder above: cd ..

1. Make sure your master branch is up to date
   1. Go to the master branch: git checkout master
   2. Update your master branch: git pull origin master

2. Create a new branch (give it a useful name): git checkout -b my_awesome_task
   1. Give your branch a sensible name about what you are working on
   2. Add your name or initials to branch name

3. Go code and write commits!
   1. git add <my file>
   2. git commit -m 'look at all this cool stuff'
4. Push your branch git push origin my_awesome_task

5. Issue a pull request
6. Have someone (maintainers) review your code
7. (Maintainers) merge the pull request
   1. You can also checkoff a box that will also delete the branch on the remote
   2. Or delete the branch manually under the branches view
8. Go back to master: git checkout master
9. Pull down your merged code: git pull origin master
10. Delete your branch: git branch -d my_awesome_task (note it is a **lower case** d)
11. Clean up your branches: git fetch --prune