# Git/Github/Docker Documentation (mostly cmd line commands)

## List of Headers that need to be modified (actions needed)

GIT CMDS – There are a couple of Git Cmds missing from mannaul write up. Also there could be better explanations of what is going on. Recap this with 21Saturday Morning summary and manual write up.

GIT CMDS – put in why or meaning of the .gitIgnorre

GIT CMDS – add in why or meaning of the ssh key method

## CRUD

CRUD stands for **Create, Read, Update, and Delete**. These are the four basic operations you can perform on data in a database. Here’s a quick overview of each operation:

##### Create:

Adding new records to the database.

* + Example: Adding a new user to the users table.

**Ruby**

User.create(name: "John Doe", email: "john@example.com")

##### Read:

Retrieving data from the database.

* + Example: Fetching all users from the users table.

**Ruby**

User.all

##### Update:

Modifying existing records in the database.

* + Example: Updating a user’s email address.

**Ruby**

user = User.find\_by(name: "John Doe")

user.update(email: "john.doe@example.com")

##### Delete:

Removing records from the database.

* + Example: Deleting a user from the users table.

**Ruby**

user = User.find\_by(name: "John Doe")

user.destroy

##### Summary

CRUD operations are fundamental to interacting with databases and are used in almost every application that deals with data. They ensure that you can manage your data effectively and efficiently.

## General Notes About Docker Image, Container, Repository

The docker buildx build command is part of Docker Buildx, which is an extension of the traditional docker build command. It provides advanced features for building Docker images, particularly useful when working with multi-platform images, custom build contexts, or using advanced caching techniques.

The$(pwd) or ${PWD} or %cd% syntax maps the current directory to the /workspace directory inside the container.

Windows PowerShell 🡪 docker run -it -p 3000:3000 -v ${PWD}:/workspace sbrady/cs3710

* **docker run**: Start a new container from an image. You only need to do this once unless you want to start all over with a new container
* **-it**: Interactive terminal for interacting with the container.
* **-p 3000:3000**: Exposes port 3000 on the host and maps it to port 3000 in the container.
* **-v $(pwd):/workspace**: Mounts the current directory from your host machine to /workspace in the container, allowing file synchronization.
* **your\_dockerhub\_username/cs3170\_rails**: The Docker image to use.

**Above created the container**

Image: The static blueprint for a Docker container. It defines the filesystem and software environment but does not change after being created.

Container: The live, running instance of a Docker image. It can be modified during its runtime and provides an isolated environment for executing applications.

To run the container using PowerShell goto dir that was mapped. M:\Web\_app\_class\Mapped\_Docker\_Container using cd… , ls , and pwd commands. Once in dir here are the commands to use for docker

## Docker CMDS and Definitions

The docker ps -a command lists all containers on your system, including those that are currently running, stopped, or exited.

The docker stop <container\_id\_or\_name> command is used to stop a running container. You can specify either the container ID or the container name docker

stop my\_nginx docker stop a1b2c3d4e5f6

The docker restart <container\_id\_or\_name> command is used to restart a running or stopped container. This command stops the container and then starts it again.

The docker exec -it <container\_id\_or\_name> /bin/sh command allows you to start an interactive shell session inside a running container. This is useful for debugging or inspecting the container’s environment. The -it flags stand for interactive terminal, which means you’ll be able to interact with the shell as if you were logged into the container’s terminal.

To check the version of Ruby installed on your system, you can use the ruby -v command. 🡪

ruby 3.3.5 (2024-09-03 revision 123456) [x86\_64-linux]

To exit the shell and return to your command line, simply type exit and press ENTER. [This will terminate the shell session inside the container and bring you back to your original command line prompt](https://www.digitalocean.com/community/tutorials/how-to-use-docker-exec-to-run-commands-in-a-docker-container)

A stack refers to the tools, frameworks, technologies, and languages used to develop software applications.

The docker attach command allows you to connect to a running container’s main process. Here’s how you can use it: docker attach containerName

docker attach my\_container

Checking Running Containers: If you’re unsure of the container name, you can list all running containers with:

docker ps

## GIT CMDS – setting/connecting to remote Repository, Creating Branches and Navigation

git status: This command shows the state of the working directory and the staging area. It lets you see which changes have been staged, which haven’t, and which files aren’t being tracked by Git.

git branch: This command lists all the branches in your repository and shows the current branch you’re on. You can also use it to create, rename, or delete branches.

git log: This command shows the commit history for the repository. It displays a list of commits, including the commit hash, author, date, and commit message.

git add . : This command stages all the changes in the current directory and its subdirectories for the next commit. The . is a shorthand for “all files”. (don’t forget the period after add)

git commit -m "message": This command commits the staged changes to the repository with a message describing what the commit does. The -m flag allows you to include the commit message directly in the command.

Example: git commit -m "Initial commit"

Example if no changes but want different “message”🡪 git commit –allow-empty -m “different message”

you’re trying to create an annotated tag with a message 🡪 git tag -a v0.0 -m "GE02 v0.0"

Here’s what each part does: -a v0.0: Creates an annotated tag named v0.0 🡪 -m "GE02 v0.0": Adds the message “GE02 v0.0” to the tag. After creating the tag, you can push it to the remote repository with:

git push origin v0.0

This will ensure your tag is also available in the remote repository.

create a new branch named branchTest. If you want to switch to this new branch, you can use the following command:

git checkout branchTest

Or, if you’re using Git version 2.23 or later, you can use the switch command:

git switch branchTest

This will move you to the branchTest branch, where you can start making changes independently of the main branch.

1. git push --set-upstream origin branchTest
2. **What This Command Does**:
   * git push: This command is used to upload your local repository content to a remote repository.
   * --set-upstream: This flag sets the remote branch as the default upstream branch for the local branch. This means that future git push and git pull commands will know which remote branch to interact with.
   * origin: This is the name of your remote repository. By default, origin is the name given to the remote repository you cloned from.
   * branchTest: This is the name of the branch you are pushing to the remote repository.
3. **Outcome**:
   * The branch branchTest was created on your remote repository.
   * The local branch branchTest is now set to track the remote branch branchTest. This means that when you run git push or git pull while on branchTest, Git will know to push to or pull from origin/branchTest.

By pushing the new branch to the remote repository, you ensure that your work on branchTest is backed up and can be accessed or collaborated on by others.

What This Command Does:

git push: This command is used to upload your local repository content to a remote repository.

--set-upstream: This flag sets the remote branch as the default upstream branch for the local branch. This means that future git push and git pull commands will know which remote branch to interact with.

origin: This is the name of your remote repository. By default, origin is the name given to the remote repository you cloned from.

branchTest: This is the name of the branch you are pushing to the remote repository.

Outcome:

The branch branchTest was created on your remote repository.

The local branch branchTest is now set to track the remote branch branchTest. This means that when you run git push or git pull while on branchTest, Git will know to push to or pull from origin/branchTest.

## Ruby on Rails

#### Running rails db:migrate

in a Ruby on Rails application applies any pending migrations to your database. Migrations are a way to alter your database schema over time in a consistent and easy way.

Here’s a quick rundown of what happens when you run this command:

1. **Pending Migrations**: Rails checks for any migration files in the db/migrate directory that haven’t been applied yet.
2. **Schema Changes**: It applies these migrations in the order they were created, making the necessary changes to your database schema.
3. **Schema Version**: Rails updates the schema\_migrations table to keep track of which migrations have been run.

##### Schema

In the context of databases, a **schema** is the structure that defines the organization of data. It includes the definitions of tables, columns, data types, relationships, indexes, and other elements that make up the database. Essentially, the schema is a blueprint of how the database is constructed.

Here are some key aspects of a schema:

1. **Tables**: These are the basic units of storage in a database, where data is stored in rows and columns.
2. **Columns**: Each table has columns that define the type of data that can be stored (e.g., integers, strings, dates).
3. **Relationships**: Schemas define how tables relate to each other, such as through foreign keys.
4. **Indexes**: These are used to speed up data retrieval operations.
5. **Constraints**: Rules applied to data in the database to ensure accuracy and integrity (e.g., unique constraints, not null constraints).

In Ruby on Rails, the schema is typically managed through migrations, which allow you to evolve your database schema over time.

##### Migration

In Ruby on Rails, a **migration** is a way to alter your database schema over time in a consistent and easy manner. Migrations are used to create tables, add columns, change data types, and more. They are written in Ruby and stored in the db/migrate directory of your Rails application.

Here are some key points about migrations:

1. **Version Control**: Migrations allow you to version control your database schema, making it easier to track changes over time.
2. **Reversible**: Migrations can be rolled back, allowing you to undo changes if something goes wrong.
3. **Consistency**: They ensure that changes to the database schema are applied consistently across different environments (development, test, production).

#### Rails Server

1. **Restart the Server**:
   * You can restart your Rails server by stopping it (usually with Ctrl+C in the terminal) and then starting it again with the command:
   * rails server
2. **Access the URLs**:
   * Open your web browser and navigate to http://localhost:3000/students and http://localhost:3000/.
3. **Create a New Student**:
   * On the http://localhost:3000/students page, there should be a link or button to create a new student. Click on it and fill out the form to create a new student.
4. **Check the Server Logs**:
   * After starting the server and performing the actions above, look at the terminal where your server is running. The logs will display the HTTP requests being made.

**Routes, Resources, and HTTP Methods**

In a typical Rails application, the routes for the students resource might look like this in your config/routes.rb file:

**Ruby**

resources :students

This generates the following RESTful routes:

* **GET /students**: Index action to list all students.
* **GET /students/new**: New action to display a form for creating a new student.
* **POST /students**: Create action to save a new student.
* **GET /students/:id**: Show action to display a specific student.
* **GET /students/:id/edit**: Edit action to display a form for editing a student.
* **PATCH/PUT /students/:id**: Update action to update a specific student.
* **DELETE /students/:id**: Destroy action to delete a specific student.

**Example Log Output**

When you visit http://localhost:3000/students, you might see a log entry like this:

Started GET "/students" for 127.0.0.1 at 2024-09-25 18:16:56 -0700

Processing by StudentsController#index as HTML

Rendering students/index.html.erb within layouts/application

Student Load (0.2ms) SELECT "students".\* FROM "students"

Rendered students/index.html.erb within layouts/application (Duration: 5.3ms | Allocations: 1234)

Completed 200 OK in 10ms (Views: 8.0ms | ActiveRecord: 0.2ms | Allocations: 2345)

When you click the link to create a new student, you might see:

Started GET "/students/new" for 127.0.0.1 at 2024-09-25 18:17:10 -0700

Processing by StudentsController#new as HTML

Rendering students/new.html.erb within layouts/application

Rendered students/new.html.erb within layouts/application (Duration: 3.5ms | Allocations: 987)

Completed 200 OK in 7ms (Views: 5.0ms | ActiveRecord: 0.1ms | Allocations: 1234)

Useful commands.

For the

For the docker command: docker exec -it <container\_id\_or\_name> /bin/sh, Changing /bin/sh to /bin/bash allows you to use up arrow to re-enter commands.

If you accidentally exit the program without stopping it from running, go to tmp, then pids, and then the server file. You will find a pid for the server that’s still running. In the command line, run kill -9 [pid], with the pid being the server pid, and then you can delete the server file.

##### M04

At the start, in student controller, add the following code

If @search\_params.emtpy?

@students = @students.none

End

This makes the program look to see if the search function is empty. If it is empty, then it will not return any students.

After that, comes the students in a neat list. To make it a row, create a new file in the path app/views/students/ called \_student\_row.html.erb file to

Make the file be a row class and have all the information be in col classes. This makes it so that the class getting called connected to each other horizontally, which makes it a nice clean line. Below is my code to show how this works.

<div id="<%= dom\_id student %>" class="row">  
  
 <div class="col-md-2">  
 <p>  
 <strong>First name:</strong><br>  
 <%= student.first\_name %>  
 </p>  
 </div>  
  
 <div class="col-md-2">  
 <p>  
 <strong>Last name:</strong><br>  
 <%= student.last\_name %>  
 </p>  
 </div>  
  
 <div class="col-md-2">  
 <p>  
 <strong>Major:</strong><br>  
 <%= student.major %>  
 </p>  
 </div>  
  
 <div class="col-md-2">  
 <p>  
 <strong>Expected graduation date:</strong><br>  
 <%= student.expected\_graduation\_date %>  
 </p>  
 </div>

Next, would be the buttons at the end for show, edit, and delete. Make it a class button group, which will have it as a group of 3 buttons. This is the code for it:

<div class="col-md-2">  
 <strong>Actions:</strong><br>  
 <div class="btn-group" role="group" aria-label="Student actions">  
 <%= link\_to "Show", student, method: :get, class: "btn btn-primary" %>  
 <%= link\_to "Edit", edit\_student\_path(student), class: "btn btn-warning" %>  
 <%= link\_to "Delete", student, method: :delete, data: { confirm: "Are you sure?" }, class: "btn btn-danger" %>  
 </div>  
</div>

The <strong> makes it say actions at the top of the page. The class=”btn-group” means that its 3 side by side buttons. It being three link\_to’s mean that they can be directed to links, and the the :btn btn-primary” and other two are what give it its color.

After that, we want the page to show the row of students when searching for students, but the full student profile when we show a student. To have this done, we change the code in app/views/students/index.html.erb to:

<div id="students">  
 **<%** @students.each do |*student*| %>  
 <%= render('student\_row', student: *student*) %>  
 **<%** end %>  
</div>

This will render the student row first, and then the student by itself.

Next, we want to update the search form so that we can search for people graduating before or after a certain date as well as search for everyone. To do this, we need to make 2 more rows for the search bars as well as a new button. A way to implement this is the code below:

<%= form\_with(url: students\_path, method: :get, local: true, class: "form-inline") do |*form*| %>  
 <div class="form-group mr-3">  
 <%= *form*.label :major, "Select Major", class: "form-label" %>  
 <%= *form*.select :major, options\_for\_select(Student::VALID\_MAJORS), { prompt: "Any Major" }, name: 'search[major]', class: "form-control" %>  
 </div>  
  
 <div class="form-group mr-3">  
 <%= *form*.label :date\_condition, "Search For:", class: "form-label" %>  
 <%= *form*.select :date\_condition, options\_for\_select([['Before', 'before'], ['After', 'after']]), { prompt: 'Select Condition' }, name: 'search[date\_condition]', class: "form-control" %>  
 </div>  
  
 <div class="form-group mr-3">  
 <%= *form*.label :expected\_graduation\_date, "Select Expected Graduation Date", class: "form-label" %>  
 <%= *form*.date\_field :expected\_graduation\_date, name: 'search[expected\_graduation\_date]', class: "form-control" %>  
 </div>  
  
 <div class="form-group mr-3">  
 <%= *form*.submit "Search", class: "btn btn-primary" %>  
 </div>  
  
 <div class="form-group">  
 <%= link\_to "Search All", students\_path(search: { major: '' }), class: "btn btn-secondary" %>  
 </div>  
**<%** end %>

The first one is to select the major, the second one is new and lets you choose if its before or after the selected graduation date. The third one is to select the graduation date, and it’s having you choose a date.