Version Control

What is Version Control?

Version Control is the practice of ***tracking and managing changes*** to software code. The version control system will keep track of every modification that is made so that if any mistakes are made, the engineers can return back to a previous working version. Since version control systems keep track of changes, engineers can work on separate parts of the project and combine their modifications seamlessly.

Git

Git is a version control system that is commonly used by engineers to keep track of code versioning in code repositories, a collection of files. The following Git commands will be run from the terminal.

Introduction to Git (from Github)
What is Git? (from the documentation)

Staging & Committing

https://git-scm.com/about/staging-area

Setup

Install Git
Configuration:
The following commands set the name and user name credentials to be used for commits.

```
git config -global user.name "[name]"
git config -global user.email "[email address]"
```

Check your settings using

```
git config --list
```

Authentication

Recording Changes

https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

Branching Basics

https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Merging Branches

https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Workflow Example

```
git init "[repo name]"
git checkout -b [branch name]
git status
git add [file name]
git add -m "[summary message of changes]"
git push origin [branch name]
```

More

- Viewing Commit History
- Undoing Things

Much much more on git (from the documentation)

https://git-scm.com/book/en/v2

git commands cheat sheet

Github

Quickstart Guide

Project

- an introduction to git with BitBucket
- git branching playground

Sources

What is version control? https://www.atlassian.com/git/tutorials/what-is-version-control
What is Git? https://git-scm.com/
Git repository https://www.geeksforgeeks.org/what-is-a-git-repository/
Git commands https://dzone.com/articles/top-20-git-commands-with-examples
Git basics, the only introduction you'll ever need
https://areknawo.com/git-basics-the-only-introduction-you-will-ever-need/

Pull Request & Peer Review Guide

Prerequisites

Please install the following

- vscode
- ESlint vscode extension (dbaeumer.vscode-eslint)
- Prettier - Code formatter vscode extension (esbenp.prettier-vscode)

Run **`yarn install`** to make sure you have prettier and eslint. If for some reason that doesn't work, you can run **`npm install prettier`** and then **`npm install eslint`**
Also make sure you have *format on save* for vscode enabled by following the instructions here

When creating pull requests...

Things you should do to make it easy for the reviewer to understand your changes:

- formatting: **make sure you have vscode, ESlint, and prettier installed, and have auto-formatted before committing**
- keep PRs small
- create a branch for your individual changes, which will be merged into a branch for your user story
- commits should a message format of **NEU-[Ticket Number]: [Description of changes]**
- summarize what you're trying to accomplish in the pull message
- add comments to your PR to help guide the reviewer
    - examples: a bunch of lines are from re-indenting, one file is particularly important, one file is tightly coupled with another file
- **screenshots if applicable**
- a test plan: steps other people should take to verify that your change is correct or intended. For example, this could be in the form of unit tests, or steps to reproduce a bug that is fixed by this diff.

PR Template

```
# Description

## Trello Card: <!-- insert link here -->

## Screenshots

<!-- Visually communicate the updated functionality here -->

## Changes
```

- <!-- Change 1 -->
- <!-- Change 2 -->

Pre-merge Checklist:

<!-- Put an 'x' instead of a space in the boxes that are true -->

- [ ] I have gone through the [PR Guide checklist](https://docs.google.com/document/d/1QAw1m5MNT6BDDWemUp_ELLelnIyyU8QxKm9nuDaVmXk/edit#)
- [ ] I have auto-formatted before committing my changes
- [ ] I have performed a self-review of my own code
- [ ] I have commented my code in hard-to-understand areas
- [ ] My changes generate no new warnings
- [ ] I have added tests that prove my fix is effective or that my feature works
- [ ] New and existing unit tests pass locally with my changes

When reviewing pull requests...

Half the battle for reviewing PRs is understanding what the PR is trying to achieve, the approach that's being taken, what's going on, and how the files all fit together.
It's your job to suggest improvements that make the code higher quality, align more with established patterns, more efficient, and more scalable, in a way that encourages growth!

Before code can be merged...

Both pull request creators and reviews should run through the following checklist:

- simplicity & readability
    - PR title and description are representative of what the changes actually do
    - simplify boolean logic
    - separation of concerns: logic is broken down into many helper functions that accomplish just one thing
    - conditionals aren't hecka nested
    - comments when necessary (your code should also be self explanatory)
    - names are semantic
    - no "any" types
- verbosity
    - follows DRY (don't repeat yourself)
    - no unreachable or unused code
    - uses shorthand syntax when possible and meaningful
- cleanliness (should mostly be taken care of by prettier)
    - aligned indentation

    - good spacing
    - semicolons where they should be
    - no new TODOs
    - no console.logs
    - no unused imports
- efficiency
    - the algorithmic complexity is reasonably efficient (no brute force)
    - asynchronous behavior is reasonably efficient
    - things that don't need to be inside components have been taken out so they don't get re-computed on re-renders
- scalability
    - "magic numbers" are denoted as constants with SCREAMING_SNAKE_CASE
    - helper functions are pure:
        - deterministic: always return the same output given the same input
        - no side-effects: do not change states outside of its scope
    - create custom hooks when applicable
    - tests cover edge cases and prevent bugs
    - doesn't reinvent the wheel: makes use of utilities that are already available (from the codebase, language, or packages)
    - patterns being used are consistent with existing patterns, or better?
- it works as expected
- there are no linting errors

PR Workflow Example

- Go to your branch in github
- Click on compare and pull request
- Add description (ticket being addressed, what you did, what it does)
- Request reviewers (at least 2)
- Go over changes requested
    - Request changes/leave comments upon reviewing files changed when you're requested as a reviewer
- Merge to master with all approvals