Week 2 - Teamwork with Git

- If you aren't already familiar with using git in a team environment, have a look at the lecture on Git Teamwork.

Congifuring your git identity

To configure your git to match your chosen username and email, do the following:

- `git config --global user.name "<user-name>"`
- `git config --global user.email johndoe@example.com`

Setting up a GitHub repository

- Sign up or Log in to GitHub

Setting up a remote repository

- **One member** of your team should make a new repository
  - On the top right of the GitHub webpage, you should see a plus icon
  - Click on that and then click `New Repository`
  - Choose a repository name
  - If you want to add a description, feel free to do so. This will just put the description in the README. You can always change/add this to the README later.
  - **Make sure that you make it private**
  - Tick to add a README, .gitignore (choose the Node template) and license (choose the MIT license)
- Add all the group members to the repository
  - Click the `Settings` icon
  - Click the `Collaborators` icon
  - Click the green `Add people` icon.
  - Add members through their git usernames, full names or email.

Cloning your remote repository

- **All members** of the team should clone the repository
  - Click the green `Code` icon
  - Copy the repository link
  - **With SSH keys**
    - If you would like to set up your SSH key, then follow this guide to generate a new ssh key (if you don't have one already)
    - Then this guide to add your ssh key to Github
  - **With HTTPS**
    - GitHub now requires 2 Factor Authentification for HTTPS URLs
    - So you will need to generate a Personal Access Token
    - Use the Personal Access Token in-place of your password when the command line prompts you for your password entry
  - Open up a terminal
  - Navigate to where you would like to store your directory locally
  - Use the command `git clone <repository-link>` to clone the repo
  - `<repository-link>` is the link that you just copied

At this point, all members should have a local copy of the repository that only includes (1) a .gitignore file, (2) a LICENSE file and (3) a README.md file

Some things to note:

- This is your **personal local copy** and what is on GitHub is your **shared remote** copy.
- If you make changes to your local copy, no one else will see those changes until you `push` them to your remote repository.
- Then to get the changes, the other team members can `pull` those changes from the remote repository.
- We will step through this process now :)

Making changes to the repository

1. Make a new file called `style-toggle.html`
2. Add the following code:

```
<!DOCTYPE html>
<html>
    <head>
        <title>INFO30005 Week 2</title>
    </head>
    <body>
        <p id="text">Hello World!</p>
        <button onClick="document.getElementById('text').style.color = 'red'">Person 1</button>
    </body>
</html>
```

- Understanding this code isn't so important right now. But it's relatively simple, so let's go through it together :)
- `<!DOCTYPE html>` specifies to the browser that the document type to expect is HTML 5
- `<html>` is the root (top-level element) which includes all the other HTML elements.
- `<head>` makes a element to include metadata. The metadata is about the HTML document but is not actually displayed.
- `<title>INFO30005 Week 2</title>` specifies the document title as `INFO30005 Week 2`. This is just for the metadata since it is enclosed between `<head></head>` tags. This is usually shown in the brower's title bar or on tabs.
- `</head>` closes the metadata element. Therefore the proceeding code is outside of the metadata.
- `<body>` makes an element to contain all the content for all the content of the HTML document, such as headings, paragraphs, button, etc.
- `<p id="text">Hello World!</p>` defines a paragraph element with the content 'Hello World!'. The `id="text"` in the p tag gives a unique id to that element. This allows it to be referenced in the future (see the following line)
- `<button onClick="document.getElementById('text').style.color = 'red'">Person 1</button>` is the most interesting part.
    - This makes a button element with the text 'Person 1'.
    - The onClick specifies an event. That is, it specifies that when a user clicks this button, the following JavaScript code should be executed.
    - The code that gets executed is: `document.getElementById('text').style.color = 'red'`
    - So when the button is clicked, the document finds the element with the `id="text"`.
    - It then changes the color of that element to red
    - Note that it is usually not good practice to inline Javascript code like this. This is something that we'll handle differently in future weeks.
- `</body>` and `</html>` closes the body and html elements respectively.
- Have a go trying out the code for yourself!
    - You can either open the file, which should take you to your default browser to view the code.
    - Or you can just right click on the file in VS Code and select 'Preview Code' option
    - Click the button and see what it does :)

Push the file to the repository

- **One member** add, commit and push the changes to the remote repository:
    - `git add style-toggle.html`
    - `git commit -m "add style toggle"`
    - `git push`

Branching (Everyone)

- So far, your team only has a `main` branch. Make you sure you have pulled the latest version.
    - `git pull`
- Create a branch `<your-name>` from the main branch.
    - `git checkout -b <your-name>`
- On that branch, edit the HTML file to add a new button with your name that changes the "Hello World!" text color to a color of your choice.
- Add, commit and push your new branch to the group repo on GitHub. I am going to leave out the commands for you to practise yourselves.
    - Note though: when you push you will need:
    - `git push --set-upstream origin <branch-name>`
    - This makes a branch with that name in the remote repository (origin) and makes your local branch track changes in that remote branch.
- Go to GitHub and you will see the option to 'Compare and Pull Request'.
- A Pull Request asks others to review your changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss the potential changes with your team and make more changes before your branch is merged into the main branch.
- Create the pull request. Then get another team member to view your pull request. If they are happy with it, they should merge the pull request. Otherwise leave a comment explaining their concerns with the changes.

Merging everyone's changes (by resolving conflicts)

- The first person to merge their pull request won't have any merge conflicts. The person who's most experienced with git can make this merge.
- Those who merge subsequently will have merge conflicts. You will need to resolve the conflicts locally, add, commit and push them before you are able to merge your pull request.
    - Go back to your main branch `git checkout main`

    - Pull in the remote changes `git pull`
    - Go back to your feature branch `git checkout <branch-name>`
    - Merge in the changes that are now in the main branch `git merge main`
    - Go to your style-toggle.html file and click accept both changes.

  We may not always want to accept all changes. This depends on the change.

    - Add the changes `git add style-toggle.html`
    - Commit the changes `git commit -m "description"`
    - Push the changes `git push`
- Now go to GitHub and merge the pull request :)

Going through the project spec

- If your team is comfortable with using git collaboratively or you have finished the previous set of tasks early, then start working through the project spec as a team.
    - You can find the project documents on the LMS
    - Start to work through the spec.
    - Try to brainstorm how the webapp will look and work.
    - What are the important design and architectural decisions that you need to make?
    - What are some technical challenges that might pop up?
    - What are some ideas to handle these techincal challenges?