Version Control (GIT & GITHUB)

- how to fork a repository on github
- Log into GitHub: Make sure you're logged into your GitHub account.
- Navigate to the Repository: Go to the GitHub repository that you want to fork. You can do this by entering the repository's URL in your browser or by searching for the repository on GitHub.
- Fork the Repository:
    - In the top-right corner of the repository's page, you'll see a button labeled "Fork". Click on this button.
    - GitHub will then prompt you to select where you want to fork the repository. Choose your own user account as the destination.
    - Wait for the Fork to Complete: GitHub will create a copy of the repository in your account. Depending on the repository's size and your internet connection, this process might take a few moments.
- Access Your Forked Repository:
    - After the forking process is complete, you will be redirected to your own copy of the repository within your GitHub account.
    - The URL will reflect that the repository is under your account, not the original owner's.
- Clone Your Fork:
    - To work with the code on your local machine, you need to clone your forked repository. Click the green "Code" button on your forked repository's page.
    - You can copy the HTTPS or SSH URL of the repository. Use the git clone command followed by the copied URL in your terminal to clone the repository to your local machine. Use the cd command to navigate to the directory where you want to clone the repository. For example, cd Documents will navigate to the "Documents" directory.
      ```
      git clone repository_url
      ```
      Replace repository_url with the URL you copied.

- **Make Changes and Push:**
    - You can now make changes to the code on your local machine using a code editor or IDE.
    - After making changes, use git add, git commit, and git push to commit your changes to your forked repository on GitHub.

- **Create Pull Requests:**
    - If you want to contribute your changes back to the original repository, you can create a pull request. This is how you request the original repository's owner to consider integrating your changes.
    - Navigate to the original repository and click on the "Pull Requests" tab. Then click on the "New Pull Request" button and follow the instructions to submit your changes.

- ## **Branching (creating and switching between branches)**
    - **Check Current Branch:**

Before creating a new branch, it's a good practice to make sure you're on the branch from which you want to branch off.

```
git branch
```

The current branch will be indicated by an asterisk (*****).

- **Switch to the New Branch:**
  To switch to the new branch you just created, use the **git checkout** command followed by the branch name:
  ```
  git checkout feature-branch
  ```
- Alternatively, you can combine branch creation and switching using the **-b** flag with the **git checkout** command:
  ```
  git checkout -b feature-branch
  ```

- **Work on the New Branch:**
  Now you are on the new branch, and any changes you make will only affect this branch. You can work on new features, bug fixes, etc.

- **Commit Changes on the New Branch:**
  Use the standard Git workflow to make changes, stage them with (**git add .**) for all files **(git add myfile.txt)** for a specific file**,** and commit them with **git commit**. `git commit -m "Your commit message here"`
  Use **git log** to see the commit history, showing each commit details.

- Pushing Commits:
  After committing changes, your commits are still only on your local machine. To update the remote repository on platforms like GitHub, you need to use the **git push** command:
  ```
  git push origin branch-name
  ```
  Replace **branch-name** with the name of the branch you're working on.

- **Unstage Changes:**
  If you accidentally staged changes you didn't intend to, you can **unstage** them using git restore: `git restore --staged myfile.txt`

- **Editing the Last Commit (if needed):**
  If you need to make changes to the last commit (e.g., fixing the commit message), you can use the **git commit --amend** command:

- **Switch Between Branches:**
  To switch back to the main branch (or any other branch), you can use the **git checkout** command followed by the branch name:
  ```
  git checkout main
  ```

- **Merge Changes:**
  Once you're done with the changes on your feature branch, you can merge them back into the main branch or another target branch. This is usually done with a pull request or the **git merge** command.

Note: Assuming you want to merge changes from a feature branch (let's call it "feature-branch") into the main branch:

- **Switch to the Target Branch:**
  Before merging, switch to the branch you want to merge the changes into. For example, if you want to merge into the main branch:
  `git checkout main`

- **Pull Latest Changes (Optional):**
  It's a good practice to pull the latest changes from the remote repository to ensure you have the most up-to-date version of the branch before merging. Use: `git pull origin main`

- **Merge Changes:**
  Now, merge the changes from the source branch (feature-branch) into the current branch (main) using the git merge command: `git merge feature-branch`

- **Resolve Merge Conflicts (if needed):**
  If Git encounters conflicting changes (changes that occurred in both branches and can't be automatically merged), it will pause and ask you to resolve the conflicts manually. Open the affected files, make the necessary adjustments, and then stage and commit the resolved files.
- Commit the Merge:
  After resolving any conflicts, commit the merged changes with a commit message describing the merge: `git commit -m "Merge feature-branch into main"`

- Push Merged Changes:
  Finally, push the merged changes to the remote repository to make them available to others: `git push origin main`
-

- **Delete Branch (Optional):**
  After merging changes from the feature branch, you might want to delete it. To delete a branch, you can use the **git branch -d** command followed by the branch name:
  `git branch -d feature-branch`
-