**GIT**

**Basic Git Workflow**

* Git --- Is a software that allows you to keep track of changes made to a project over time. It works by recording the changes you make to a project, storing those changes, then allowing you to reference them as needed.
* git init --- Turns the directory into a Git project
  + The word ‘init’ means initialize.
  + The command sets up all the tools Git needs to begin tracking changes made to the project
* Git Workflow --- Can be thought of as having three parts
  + A Working Directory: where you’ll be doing all the work: creating, editing, deleting and organizing files.
  + A Staging Area: where you’ll list changes you make to the working directory
  + A Repository: where Git permanently stores those changes as different versions of the project.
  + In Git, we save changes with a ‘commit’.
* Git Status --- Used to check the status of the working directory.
  + Untracked Files: Means that Git sees the file but has not started tracking changes yet.
* Git add --- Used to add a file to the staging area
  + Necessary in order for Git to start tracking the file.
  + **EX** git add filename
  + Changes to be committed: The file was added to the staging area.
* git diff --- Used to check the difference between the working directory and the staging area
* git commit --- The last step in our Git workflow. Permanently stores changes from the staging area inside the repository.
  + Use the option -m to include a message.
    - Must be in quotation marks
    - Written in present tense
    - Should be brief (50 characters or less).
* git log --- Used when you need to refer back to an earlier version of a project.
  + Commits are stored chronologically in the repository and can be viewed with ‘git log’

**Basic Branching**

* Git allows us to create branches to experiment with versions of a project. The branches will have no effect on the master branch until you’re ready to merge them.
  + **EX** git branch
    - This will tell you which branch you are on. In the output, the \* is showing you what branch you’re on.
  + **EX** git branch new\_branch
    - This will create a new branch.
    - Be sure to name your branch something that describes the purpose of the branch.
* git checkout
  + **EX** git checkout branch\_name
    - Used to switch from the master branch to a new branch.
    - Once you switch branches, you are now able to make commits on the branch and have no impact on ‘master’
      * You can continue your workflow while ‘master’ stays intact.
* git merge
  + **EX** git merge branch\_name
    - Used to merge a branch with the master branch
    - This is the giver branch, since it provides the changes.
    - Master is the receiver branch, since it accepts those changes.
    - You also have to be set on the master branch when you merge.
* Merge conflict
  + Sometimes if you try to merge a file to the master file, if the same line is changed on both files, it will cause a merge conflict.
  + Once this happens, git will automatically show the two versions in the text editor. Delete the version that you don’t want, then add it to the staging area and make the commit.
* Delete branch
  + Git branches are usually used to work on a new project feature, but the end goal is to merge that feature into the master branch. After the branch has been integrated into ‘master’, it has served its purpose and can be deleted.
    - EX git branch -d branch\_name
      * This will delete the specified branch from your Git project.
* echo "# practice2" >> README.md
* git init
* git add README.md
* git commit -m "first commit"
* git branch -M main
* git remote add origin https://github.com/sama4549/practice2.git
* git push -u origin main