Week 1 Assignment

Ref URL - https://learngitbranching.js.org/

1. Git-hub:
    a. Basic usage using the CLI
        i. Setup a Local Repository
        ii. Setup a Remote Repository
        iii. Create Local branches (Feature branch, Dev. branch, QA Branch, Master / Prod. Branch, Delivery Branches)
        iv. Create Remote branches
        v. Add files, Make changes to existing files, Add folders, Remove folders, remove files
        vi. Check-in, Stage, Commit, Push files into Feature Branch
        vii. Promote code from Feature branch to Dev. branch via Pull requests
        viii. Check-out the latest code from remote branch to local branch
        ix. Explore the diff. between Checkout vs Pull
        x. Get two people to make changes to the same file, check-in & handle merge conflicts
        xi. Ensure the code is in sync. with the latest changes across all branches à from Prod. à QA à Dev. à Feature

    b. Basic usage using the GUI Client (VS Code or Github Client etc.)
        i. <Repeat all the above steps using a GUI>

    c. On the ground day to day scenarios
        i. Reset / revert one or more files to the previous state & ignore the local changes (Soft reset & Hard reset)
        ii. Stash the local changes during merge conflicts
        iii. Rebasing with all options (reword, edit, squash, fixup, exec, drop)
        iv. Git log, status & reflog

    d. Follow the assignments / ready made from git guides
        i. Git command-line interface and conventions → git help cli
        ii. A Git core tutorial for developers → git help core-tutorial
        iii. Tweaking diff output → git help diffcore
        iv. A useful minimum set of commands for Everyday Git → git help everyday
        v. A Git Glossary → git help glossary
        vi. Specifies intentionally untracked files to ignore → git help ignore
        vii. Defining submodule properties → git help modules
        viii. A tutorial introduction to Git → git help tutorial
        ix. A tutorial introduction to Git: part two → git help tutorial-2
        x. An overview of recommended workflows with Git → git help workflows

1.

a)

i)

- Create a directory to contain the project
- Go into the new directory and right click and the select git bash here
- Type **git init**

ii)

- Go to github
- Log in to your account.
- Click the new repository button in the top-right.
- Click the “Create repository” button
- Right Click and select git bash here.
- $ git remote add origin git@github.com:username/new_repo
- $ git push origin master

iii)

git branch feature
git branch dev
git branch QA
git branch prod
git branch delivery

iv)

1. create local repository of remote repository
2. go to local repository and Right Click and select git bash here.
3. git branch feature
4. git branch dev
5. git branch QA
6. git branch prod
7. git branch delivery
8. git push origin --all

v)

open git bash
-**add files**
touch file_name. file_type
-**modify files**
echo “add this to file” > file_name. file_type
-**add folders**
mkdir folder_name
-**remove folders**
rm -r folder_name

-**remove files**

rm file_name

vi) git status

git add .

git commit -m “first commit”

git push origin feature

vii) Go to the repository page on github. And click on "Pull Request" button in the repo header.

viii)

git pull

ix)

git checkout is a command used for check out (or) changing from one branch to another branch .

Ex: **git checkout “branch name”**

git pull is a command used to brings your local copy (more) up to date by merging any changes in the remote branch..

Ex: **git pull**

x)

The git log --merge command helps to produce the list of commits that are causing the conflict

cat file_name

The easiest way to resolve a conflicted file is to open it and make any necessary changesAfter editing the file, we can use the git add . command to stage the new merged content

The final step is to create a new commit with the help of the git commit command

Git will create a new merge commit to finalize the merge

xi) git pull –all

b)

i) For creating a local repository: in our Git GUI, click on "Create New Repository"**.**
Select the location you wish to store your repository in. It is important to note that the selected repository location

ii)

Go to github
Log in to your account.
Click the new repository button in the top-right.
Click the "Create repository" button

iii)

To create a new branch in git gui, choose Branch → Create

iv)

create local repository of remote repository
go to local repository and Right Click and select git bash here.
To create a new branch in git gui, choose Branch → Create
Click on push

v) same as usuals

vi) Clicking the *Rescan* button in the git gui will cause it to search out new, modified, and deleted files in the directory
To add the file for committing, click the icon to the left of the filename. The file will be moved from the *Unstaged Changes* pane to the *Staged Changes* pane. Now we can add a commit message and commit the change with the *Commit* button.

vii) Go to the repository page on github. And click on "Pull Request" button in the repo header

viii) go to "Remote" menu, then "Fetch from" option , in my case origin, and then go to "Merge Menu" and then "Local Merge.

ix)

checkout is used for check out (or) changing from one branch to another branch .
pull("Fetch from" + "Merge Menu") is used to brings your local copy (more) up to date by merging any changes in the remote branch.

x) edit the file creating conflict and Clicking the *Rescan* button in the git gui will cause it to search out new, modified, and deleted files in the directory
To add the file for committing, click the icon to the left of the filename. The file will be moved from the *Unstaged Changes* pane to the *Staged Changes* pane. Now we can add a commit message and commit the change with the *Commit* button

xi) go to "Remote" menu, then "Fetch from" option , in my case origin, and then go to "Merge Menu" and then "Local Merge.

c)

i)
when we want our repository to point to the commit with the hash that starts with 58c8e75 and forget as if commits after that ever existed. There are a few ways to achieve this.
git reset –soft 58c8e75 will remove all commits after commit 58c8e75 and will bring all changed code after that into the staging area. You don't need to use the full hash of a commit. All commits after this commit are then removed from git history.

git reset –hard 58c8e75 will remove all commits after commit 58c8e75and destroy all changed code

after that. This will also remove the changed files in the working or staging area

ii) If we make a change to the same function both locally and remotely, then when we try to pull down the remote changes, we'll run into a conflict - and git won't pull the remote changes down.
So first, we'll use git stash to stash the local changes, and then git pull to get the remote changes. To apply the stashed changed, we'll then use git stash save

iii) git rebase -i gives much more fine grained control over history modifications than a standard git rebase.

Rebasing options

- p, pick = use commit
- r, reword = use commit, but edit the commit message
- e, edit = use commit, but stop for amending
- s, squash = use commit, but meld into previous commit
- f, fixup = like "squash", but discard this commit's log message
- x, exec = run command (the rest of the line) using shell

iv) git log shows a history of all your commits for the branch you're on.

git status shows the state of the working directory and the staging area.

git reflog shows a record of your references . There is an entry each time a commit or a checkout it done