1. Introduction to terminal, Bash Script and Git

1.1 Introduction to phase 3

- Refer the class video

1.2 Bash script: Mac terminal, Git Bash and Cygwin

- **What is Bash?** It is better to it is better to define terminal, GUI and shell before jumping into Bash
    - **Graphical User Interfaces (GUI)**: A GUI is a way to command/instruct your computer operating system (OS) using graphical symbols rather. A GUI uses windows, icons, and menus to carry out commands, such as opening, deleting, and moving files. Although a GUI operating system is primarily navigated using a mouse, a keyboard can also be used.
    - **Terminal**: GUIs, allow a user to accomplish daily tasks by easily interacting with windows and icons. However, there are commands that need to be manually entered via texts to instruct the computer directly. We usually do this through terminals. Terminals, also known as command lines or consoles the "black" screen programmers use to send text commands to do things like navigate through a folder or copy a file and do more complex tasks.
    - **Shell**: When you type commands on your terminal, what you've typed must be understood before the computer executes the task. Shell is the software that interprets the commands typed on terminal so that your computer understands them and return a response to your command. There are different shells (programs/CLI/script) that we can use to manipulate our OS. **Example**: Bash (for Mac and Linus OS), Command Prompt (CMD) or PowerShell. We will focus on Bash in our course.
    - **Bash**: It is an acronym for Bourne Again SHell which is a command-line interface (CLI) and is currently the most widely used shell.
        - Bash is a type of shell (CLI) that we can use to communicate/instruct our OS. Meaning, we use Bash when we want to control our OS without having to navigate menus, options, and windows within a GUI

        - Whatever we can do with our mouse to communicate with our OS, we can do it with from our terminal by running Bash commands without the need to use our mouse to navigate through our computer
- **Why are we learning Bash and Bash script for this course?**
    - As web developers you should learn one of the shell scripts, be it Bash, Command Prompt or PowerShell for the following reasons:
        - **Great control over your operating system**: There are commands that you can't use with your mouse and keyboard, but you can achieve these commands only via shell.
            - **Example**: changing permissions on a certain file, listing all files including the hidden files, server administration for a local server setup like Apache/MySQL and most importantly running multiple commands at once (command to change your directory/folder and then listing files in that folder)
        - **The piping feature on bash is what makes it super special**: The Pipe (|) is a command that lets you use two or more commands at once such that the output from command serves as an input to the next.
            - **Example**: cd classFolder | ls -lt | head
                - cd changes the direcrory/folder
                - ls -lt, lists all files/folders in that directory in order of time created.
                - head command prints the first 10 lines of the specified files.
                - So, in the above example, we are saying "please change to the classFolder, list all files and folders in classFolder and from the listed files and folders, only show me 10 of them" in one command line
        - **There are web developer tools which operate through shell only**: There are software/application can't be download and installed in your computer. Thus, understanding shell commands will be very helpful in such cases
            - **Example**: A package from Node Package Manager (NPM) can't be downloaded and installed using your mouse. You will need to install every package manually using npm commands on your terminal window. In addition, you will need to use npm

commands to update (all packages at once or one by one), build or know the versions of your packages,

- **Better utilization of the Git Version Control tool**: You will need your shell command experiences to use the Git commands as well. There is a desktop application created to make Git use easier. However, there are a lot of Git functionalities that still require the use of Git commands from your terminal.
- **For local backend development**: The front-end languages live in the browser, so there are no extra installations required. However, any backend programming language can run on your server and there is no one language that lives in your server. Therefore, you need to install the backend language you want on your computer in order to get them running. Therefore, a backend developer (regardless of the language) will need to know a shell script during framework installation process or while setting up your computer's environment. In your case, you will need a shell scripting knowledge for setting up your own local server on Node.js
    - **Example**: initiating your local Node.js server, keeping your server running (with the nodemon command), installing modules/packages

- **What do we need to write Bash script?**
    - You need to have a terminal to write bash script on.
    - If you have a Mac computer, you can use the default built-in terminal on your Mac.
    - For Windows computers, you can applications such as GitBash and Cygwin.
        - Git Bash comes included as part of the Git for Windows package.
        - If you have not downloaded and installed Git, please download and install Git for Windows here, just like other Windows applications.
        - Once download completes, go to your downloads folder and find the included .exe file and open to execute Git Bash

1.3 The main Bash commands you must know

- **Directory**: when we discuss Bash commands, we will use the term “directory” more often. Please know that directory is another name for folder.
    - **pwd**: It prints full path of your current/working directory
    - **cd**: It will change the directory you are currently in so that you can manipulate the different files and directories in your system. It is basically going to a different folder and double clicking it
    - **ls**: Lists contents of the current directory, whether files or folders. It’s the same as you opening a folder in file explorer to see its contents
    - **mkdir**: Makes/creates a new. It is the same as you right clicking and creating a folder
    - **touch**: This is going to be the easiest way to create a new file in a current directory. It is the same as right clicking on a folder and creating a new file
    - **echo**:
        - **Using echo without option**: It prints text to the terminal window. **Example**: echo “hello world” prints “hello world” on terminal
        - **Using echo with fie name only**: It creates a file. **Example**: echo > style.css creates the style.css file
        - **Using echo with text and file name**: creates the file and writes the text in the file. **Example**: echo "Hello" > index.html, creates an index.html file and types “Hello” on the file
    - **mv**: It moves or renames directory.
        - **Example**: mv style.css CSS. This command will move the style.css file found in your current directory to the CSS folder
        - **Example**: mv style.css custom.css. This comand will rename the file style.css to custom.css It is the same as right clicking on style.css to rename it
    - **rm and rmdir**: Both remove directories. However, rmdir will only delete empty directories, whereas rm will remove directories and files regardless if they contain data or not.
        - **Example**: rm -r CSS will removes the CSS folder (this folder will not be removed by rm unless it has files or folders in it)

- Example: rmdir CSS will remove the CSS folder, whether it has files or folders in it. This is basically right clicking on a folder/file and deleting it

- **grep**: It searches for a string in groups of files from current directory
  - **Example**: grep Sam my.txt your.txt : This command is basically saying search for "Sam" string in my.txt and your.txt files found under your current directory
- **chmod**: This command is used to change the permissions of a file in a directory.
  - **File/folder owners**: Each file is associated with an owner and a group and assigned with permission access rights for three different classes of users:
    - The file owner
    - The group owners/members
    - Other users (everybody else)
  - **File/folder permission**: There are three file permission types that apply to each class of users above
    - The read permission
    - The write permission
    - The execute/running program permission
    - **Interpreting file permission listing**: **Example**: rwxrwxrwx This is interpreted to mean that
      - **The first set of rwx is for the owner**: File owner has permission to read, write and execute
      - **The second set of rwx is for the group owner of the file**: The group also has permission to read, write and execute
      - **The third set of rwx is for all other users of the file**: All other users also has permission to read, write and execute
    - **Changing file access permissions**: File access permissions can be changed by a numerical (octal) chmod specification
      - (r) read permission is given the value 4
      - (w) write permission is given the value 2
      - (e) execute permission is given the value 1
      - (-) indicates the file is a regular file/ not executable file

- **The above values are added together for any one user category**:
    - 0 means no permissions at all
    - 1 means there is permission to execute only
    - 2 means there is permission to write only
    - 3 means there is permission to write and execute (1+2)
    - 4 means there is permission to read only
    - 5 means there is permission to read and execute (4+1)
    - 6 means there is permission to read and write (4+2)
    - 7 means there is permission to read and write and execute (4+2+1). 7 means there is no restrictions on permissions. Anybody may do anything, generally, not a desirable setting
- **File permission to 664**: This means rw-rw-r--
    - **The first 6**: File owner has permission to read and write (4 + 2 + 0)
    - **The second 6**: The group owners of the file also have the permission to read and write (4 + 2 + 0)
    - **The third 4**: All other users have permission to read only (4+ 0 + 0)
- **Command to view file permission:** ls -l filename.txt
- **Command to change file permission**:
    - **Example**: chmod 660 my.txt

- **exit**: The exit command will close a terminal window/ends the session. You can just type the "exit" command while you are on your current directory
- **crtl + C**: If you don't want to close your terminal, but want to stop a running command, just use the "**Ctrl + C**" using your keyboard
- Please refer to this link for the list of further commands bash commands
    - https://www.educative.io/blog/bash-shell-command-cheat-sheet

1.4 Introduction to Git: what is Git?

- **What is Git?**
    - Git is a Version Control System. There are multiple Version Control Systems, but Git is by far and large the most popular.
    - In simple terms, Git is a software that tracks the changes you make to files, to be specific, it tracks the changes you make in your code. By tracking it is meant that Git will record every change you make in a file and saves the different versions of the file. If you want to look at a specific version of your file, you can go back and see that version.
    - Git's tracking of file changes is important when a project is done in collaboration with others by allowing changes made by different people to be merged into one document.
- **The. git directory and how does Git track changes?**
    - The .git directory: This the directory/folder where Git stores the information about your project. Without the .git directory, your project is a local project and not a Git project. Meaning, you cannot perform any git commands and the project remains untracked for any changes. The .git directory does the following:
        - Remote information (to which remote server your project is connected)
        - History of all local commits
        - Branch information (on which branch is your current project state (HEAD) pointing to)
        - All logs of all local commits you have ever made (including revert changes)
    - **Where does the. git directory come from?**
        - In case of new projects, this directory is created when you open your new project folder in terminal and type the command "git init"
        - If it is an existing project, this directory is created when you bring a copy of a repository into your local computer through the command "git clone".
- **Ways to use Git**: Git can be accessed via terminal commands or desktop app (GitHub Desktop)

1.5 Why do we use Git and GitHub? How do we create a repository on GitHub?

- **Why use Git in website/application development process?**
  - As stated above, Git is useful to anyone who writes code and tracks changes made to a file. Git simplifies the process of working with other people. Having a centrally located place (remote place such as GitHub) where you can upload your changes and download changes from other members of your team, enable you to collaborate more easily with other developers.
  - If different people are working on the same file, Git can automatically merge the changes from each contributor, without losing everyone's work.
  - **Diffing**: makes git supper powerful and efficient
    - **Diff (or short for difference)**: Diff in computation world allows programmers to look at 2 files side by side and see what differentiates them by spotting the place where a new line of code had been added.
    - Diffing is used mostly when comparing two different versions of the same file/code, meaning comparing an older with the newer version.
    - Diffing is used in Git to specifically show what has changed between two versions of the same file/code/commits/branches
- **What is GitHub?**
  - Git is software that runs locally. Your files and their history are stored on your computer. However, if you want to work on a project together with others from anywhere, you will need to store a copy of your files in a clude/remote place. That is when GitHub comes in handy.
  - GitHub is a website/online host where you can remotely store your git project/repositories so that they are accessible from anywhere in the world for anyone working on the project. It offers the distributed version control and source code management functionality of Git, plus its own features

1.6 Git workflow: cloning, editing, committing and pushing

- **Git workflow**: It is a guideline for how to use Git in a team project to ensure the team is on the same page. Even if there is no fixed standard way of interacting with Git, however, a software team must at least know where to begin when implementing Git in the workplace. No matter what, the team's size and culture are always considered when planning a Git workflow.
- **Git repository/repo**: It is a virtual location on GitHub where a user can store code, and a collection of files of various versions of a Project. Repository is nothing but a folder where files and other folders are saved remotely in cloud services like GitHub.
- **What happens in Git workflow?**
    - **Creating a repository**: Someone needs to create the central repository on GitHub if the project is a brand-new project. Here are the steps to create a new repository on GitHub:
        - **Account with GitHub required**: You need to have a GitHub account to create a remote/central repository on GitHub.
            - If you do not have a GitHub account already, click the link below to sign up. **Note**: You will need an email address to sign up
                - https://github.com/
            - Once you sign up with GitHub, log into your GitHub account
            - In the upper-right corner in your GitHub account, you will see the "new" button. Click that button to start creating your repo
            - Type a short, memorable name for your repository
            - If you want, add a description of your repository
            - Select the option to initialize the repository with a README file
                - **Why README.md file?** This file tells other people why your project is useful, how they can get started using it and what they can do with your project
    - **Cloning:** If it is not a brand-new project and there exists a Git repository already, each developer needs to create a local copy of the existing repository, a process called cloning. Here is the command you use to clone a remote repository:
        - Go to any folder of your choice on your computer where you want to store the remote repository and open your terminal

- **Type**: git clone <URL address of the central repo>
    - You can find the address of the central repo by going to your GitHub first and opening the repository. Click on “Code” green button and either copy the HTTPS or SSH URL

- **Committing**: After cloning the project locally, each developer can make changes to the files and then commit. Committing is basically saving the changes the developer made to the local repository.
    - **Type**: git commit –m “_____” (where you fill in the space with a comment)
- **Pulling**: After committing changes locally, if you just push changes to the remote repository, it is possible that another developer had already pushed updates that contain code that conflicts with yours. Git will notify the conflict between your version and the other developer’s version. Before publishing your updated version, you need to download the current version of the central repository by the process called, “git pull”. Pulling the current version of the remote repository will allow you to add your updated version on top of what everyone else has already done.
    - **Type**: git pull
    - **Merging conflicts**: Without pulling the current repository to your local computer, if you push your updated version to the central repository and if the version you have is different from that of the current version of the central repository, Git will refuse to push your changes to the central repository. This is done to avoid the overwriting of other official commits. In such cases, Git will give you a chance to manually resolve the conflict between your updated version and that of the central repository, with a process called merge conflict.
    - **Note**: Merging conflicts manually is not advisable. Rather, it is highly advisable that developers pull the current version of the repository from GitHub before pushing their updated version to the central repository.
- **Pushing**: Once the local repository has the new changes committed, these changes will need to be uploaded/pushed to the remote/central repository to share with other developers on the project.
    - **Type**: git push

- **Branching**: Branching in Git allows you to separate your work into different branches so you can work freely on your source code without affecting anyone else's work or the actual code in the main branch.
    - **Main/master branch**: This is basically the first branch created when you clone and save a repository in your local computer.
    - **Other branches**: Git allows you to create branches in addition to the main branch. To create other branches, you can either create them from the main branch or from the central remote repository.
        - **Why other branches?** Let's see this real-life scenario. Let's assume you and your co-worker are working on the same project but you both are working on different features. It is best if there is one main branch then two other branches for each of you to work. When you both finish working on your features, these feature branches can be merged into the main/master branch and accepted into the main central repository.
    - **Creating a new branch:**
        - **Type**: git branch <branchName>
    - **List all of the branches in your repository:**
        - **Type**: git branch
    - **Delete a specified branch:**
        - **Type**: git branch -D < branchName >
    - **Rename the current branch to "newBrahchName":**
        - **Type**: git branch -m < newBrahchNam >

1.7 Git workflow: forking, sending pull requests and merging

- **Forking**: A fork is a copy of a repository. Forking is basically copying any public repository from GitHub of someone to your GitHub (it is copying from remote repo to another remote repo).
- **Why forking**: Forking a repository allows you to freely make changes without affecting the original project. Usually, people want to fork a publicly available repository because they want to contribute to someone else's project or want to use someone's project as the starting point for their own.
- **What happens after forking?** Forking without the need to make changes to the forked repository is useless. The very reason of forking someone's project is because you want to

contribute to the project my making changes. Once you forked someone's repo, it will be saved in your GitHub. In order to make changes to the repo, you will need to:

  - **Clone**: You will need to clone it and make it available in your local computer so that you make changes
  - **Commit**: Make changes and commit the changes
  - **Push**: Push the changes. **Note**: pushing the changes here means, the changes will be pushed to the remote forked copy found in YOUR GitHub. It will not be pushed to the original git repo. If you want to send the changes you made to the original repo owner, you will need to send a pull request.

- **Sending pull request**: This is to tell the owner of the project that you wish to publish the changes you made into their repo. Steps to send a pull request:
  - Go to your GitHub and then to the forked project. You will see the change you just committed
  - Click on "pull request" and then click on "new pull request"
  - You will see a pull request opened with a box to write a comment/note to state what changes you made. Write your note and create a pull request
- **Merging a pull request**: Once you send a pull request to propose that changes you've made on your branch, the owner of the repo needs to accept your pull request in order for the changes to be added to the original repo. If the owner is happy with the changes, she/he will click "Merge Pull request". By default, any pull request can be merged at any time, unless the head branch conflicts with the base branch
- **How can you transfer files between your local development environment/your computer and the remote central repository?** To do that, you can use the following 3 ways:
  - **1. GitHub desktop software**:
    - You can download and use the desktop App from the link below:
      - https://desktop.github.com/
    - Here is a very short but helpful tutorial for GitHub desktop:
      - https://www.youtube.com/watch?v=ci3W1T88mzw
  - **2. Git commands from VSC terminal**:
    - The easiest option would be to use Git on VSC terminal. To be able to do that, read this document and add the add GitHub Pull Requests and Issues extension on your VSC
      - https://code.visualstudio.com/docs/editor/github

- Here is a short video on how to use Git from your VSC terminal
    - https://youtu.be/Fk12ELJ9Bww

- **3. Git commands from your computer's terminal/command line interface**:
    - **Step 1: Download and install GIT and Git Bash**
        - **For Windows computers:** The other option would be to install Git Bash because the Git command line interface/terminal comes with it. Git Bash comes included as part of the Git for Windows package. If you have not already downloaded Git, click the below link to download and install Git for Windows like other Windows applications.Then, find the included .exe file from your download folder and open to execute Git Bash.
            - https://gitforwindows.org/
            - To check if Git is installed on your computer, run this command on your terminal
                - git --version
        - **For Mac computers**: The default terminal (or zsh) comes with Git pre-installed on it, so no need to install Git Bash
    - **Step 2: Track your project with Git:**
        - Go to one of your folders in your computer that you want to track with git, open it with your terminal and initialize the git version controlling system:
            - **Command**: "git init"
            - This command will create the .git folder in the directory so that Git saves any of your changes in this folder
        - However, you want to work on a repository that already exists, just bring a copy of the remote repository into your computer:
            - **Command**: "git clone urlOfTheCentralRepo"
    - **Step 3: Configure your identity on the repository**: This is basically setting your Git username for every repository on your computer
        - **command**:
            - git config --global user.name 'Sara Ali'
        - To Confirm that you have set the Git username correctly:

        - **Command**: “git config user.name”, it will print 'Sara Ali'

- **Step 4**: **Add the files you want to track to the staging area**: Staging tells Git that you want to include the updates you made in your next commit. However, your changes are not actually recorded until you run “git commit”
    - **command**: git add [file-name] or git add .
    - To check what is added to the staging area use the status:
        - **command**: git status
    - If you want to remove a file you just added to the staging area:
        - **Command**: git rm --cached [file-name]
        - the --cached keyword removes the files from staging area, but the file will be removed from the index tracking your Git project, not from your local repository.
- **Step 5**: **Commit your change to the Git repository**
    - **Command**: git commit -m "A note to help you remember the changes you made"
    - **Note**: You must explicitly tell Git which changes you want to include in a commit (by adding these changes to the staging area) before running the "git commit" command. This is because the "git commit" command will add only the changes that are added to the staging area
    - **Note**: Using the "git commit" command only save the change you made in your local Git repository
- **Step 6**: **Using .gitignore file to identify files you don't want to be tracked by Git**: If there are files and folders you do not want Git to track, make sure to add them into your .gitignore file. These files inclide a file is not used by anyone else in your team (like your to-do list file), files with API keys/secrets, credentials, or sensitive information and dependencies which can be downloaded from a package manage How?
    - **Creating a local .gitignore file in your repository**: All you need to do for this is, create a text file and name it .gitignore (remember to include the dot at the beginning). You can mannually create this file or create it from terminal using “touch .gitignore”

- **Adding files/folders to the .gitignore file**: to and add all the files you want to be ignored in your .gitignore
    - **Ignoring a file**: just type name of the file to ignore in the .gitignore file. **Example**: todo.txt
    - **Ignoring a folder/directory**: To ignore entire directories, just by including their paths and putting a forward slash (/) at the end. **Example**: node_modules/
    - **Ignoring in group using Wildcard (*)**:
        - **Example:** *.log ignores any file ending with the .log extension
    - **Negation (!)**: Putting negation before the file name negates a file that would be ignored otherwise
        - **Example**: !example.log, this means that example.log is not ignored, even though all other files ending with .log are ignored already
    - **Comments**: Any lines that start with # are comments. **Example**: # macOS Files
  - **Note**: Once you added a file/folder in your .gitignore file, all your Git repositories will ignore the files and folders listed in the .gitignore file.
- **Step 7**: **Create a remote repository on GitHub**: How to create a repository from GitHub is discussed above. Follow those steps and create a repository
- **Step 8**: **Add the remote repo location to our local git folder**: This is to tell Git that we need a remote repo to push whatever we do with our local folder.
  - For that, you will need the address/URL of your GitHub repository. To get this address, just
    - Open the repository you just created on GitHub, and click on “code” button on green and either use the HTTPS or SSH address of the remote repo
  - Now tell git that you want to add the remote repo location to push your local changes to:

        - **Command**: git remote add origin https://github.com/yourgithubusername/repoName.git
    - To check if remote repo is added successfully:
        - **Command:** git remote
- **Step 9**: Push whatever files and changes you made locally to the remote Git repo
    - **Command**: git push -u origin master
        - If this is your first time pushing a local change to the central repo, you will be asked to provide your GitHub credentials
    - Now, go to your GitHub and open your remote repo. You will see the commit you made

- **How to use an existing central/remote repo?**
    - **Step 1**: **Clone/copy the remote repo**: **Command**: git clone <remoteRepoURL>
    - **Step 2**: **Make changes to the copy of the repo locally**
    - **Step 3: Add all changes to staging area: Command**: git add –A
    - **Step 4**: **Commit the changes**: Command: git commit –m "Type message"
    - **Step 5**: **Pull the current version of the central repo: Command**: git pull
        - It is possible that another developer had already pushed updates that contain code that conflicts with yours. To incorporate changes made by other developers, before you push your changes to the remote repoyou, you will ALWAYS need to first import their changes to your local directory by git pull command
    - **Step 6**: **Publish your changes to the remote repo: Command:** git push
- **For essential Git commands every developer should know**
    - https://dev.to/dhruv/essential-git-commands-every-developer-should-know-2fl
- **To learn more about Git, watch "The Git crash course for beginners, click below**:
    - https://www.youtube.com/watch?v=SWYqp7iY_Tc