**GIT**

1. Git is a version control system.
2. Git helps you keep track of code changes.
3. Git is used to collaborate on code.
4. In general, lines with $ in front of it is input. These are the commands you can copy and run in your terminal.

# **INTRODUCTION**

1. A popular version control system.

## **USE**

1. Tracking code changes
2. Tracking who made changes
3. Coding collaboration

## **DO**

1. Manage projects with Repositories
2. Clone a project to work on a local copy
3. Control and track changes with Staging and Committing
4. Branch and Merge to allow for work on different parts and versions of a project
5. Pull the latest version of the project to a local copy
6. Push local updates to the main project

## **WORKING**

1. Initialize Git on a folder, making it a Repository
2. Git now creates a hidden folder to keep track of changes in that folder
3. When a file is changed, added, or deleted, it is considered modified
4. You select the modified files you want to Stage
5. The Staged files are Committed, which prompts Git to store a permanent snapshot of the files
6. Git allows you to see the full history of every commit.
7. You can revert back to any previous commit.
8. Git does not store a separate copy of every file in every commit but keeps track of changes made in each commit!

## **WHY?**

1. Over 70% of developers use Git!
2. Developers can work together from anywhere in the world.
3. Developers can see the full history of the project.
4. Developers can revert to earlier versions of a project.

# **GITHUB**

1. Git is not the same as GitHub.
2. GitHub makes tools that use Git.
3. GitHub is the largest host of source code in the world and has been owned by Microsoft since 2018.

# **CREATING GIT FOLDER**

1. **mkdir** makes a new directory.
2. **cd** changes the current working directory.

# **NEW FILES**

1. **ls** will list the files in the directory.
2. Git **status** and see if it is a part of our repo.
3. Files in your Git repository folder can be in one of 2 states:  
   -Tracked - files that Git knows about and are added to the repository  
   -Untracked - files that are in your working directory, but not added to the repository

# **STAGING ENVIRONMENT**

1. One of the core functions of Git is the concepts of the Staging Environment, and the Commit.
2. **Staged files** are files that are ready to be committed to the repository you are working on.
3. A **README.md** file that describes the repository (recommended for all repositories)

# **COMMIT**

1. Adding commits keep track of our progress and changes as we work.
2. Git considers each commit change point or "save point".
3. It is a point in the project you can go back to if you find a bug or want to make a change.
4. When we commit, we should always include a message.  
   By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.
5. It is possible to commit changes directly, skipping the staging environment.
6. The -a option will automatically stage every changed, already tracked file.
7. Short status flags:  
   -?? - Untracked files  
   -A - Files added to stage  
   -M - Modified files  
   -D - Deleted files
8. Skipping the Staging Environment is not generally recommended.  
   Skipping the stage step can sometimes make you include unwanted changes.

# **HELP**

1. If you are having trouble remembering commands or options for commands, you can use Git help.
2. Different commands:  
   -git command -help - See all the available options for the specific command  
   -git help --all - See all possible commands
3. Any time you need some help remembering the specific option for a command, you can use **git command -help.**
4. You can also use --help instead of -help to open the relevant Git manual page.
5. To list all possible commands, use the **help --all** command.
6. If you find yourself stuck in the list view, SHIFT + G to jump the end of the list, then q to exit the view.

# **BRANCH**

1. A **branch** is a new/separate version of the main repository.
2. Branches allow you to work on different parts of a project without impacting the main branch.
3. When the work is complete, a branch can be merged with the main project.
4. You can even switch between branches and work on different projects without them interfering with each other.
5. Branching in Git is very lightweight and fast!
6. **Checkout** is the command used to check out a branch. Moving us from the current branch, to the one specified at the end of the command.
7. Using the -b option on checkout will create a new branch, and move to it, if it does not exist.