

Lab 1

Introduction to GIT and GitHub

Introduction

GIT is a version control system that allows us to collaborate with other developers. It also allows us to keep track of how our files changes over time. It is designed to handle everything from small to very large projects with speed and efficiency.

Version Control System (VCS): Git tracks changes to files and allows multiple people to collaborate on a project. It records the history of every change and can revert to previous versions if needed. Here are some key points about Git:

1. **Distributed System:** Unlike traditional version control systems, Git is distributed. This means every developer has a full copy of the repository on their local machine, including the entire history of changes.
2. **Branching and Merging:** Git allows users to create branches to work on different features or fixes independently. Branches can be merged back into the main codebase once the work is complete. This helps in maintaining a clean and organized project history.
3. **Speed and Efficiency:** Git is designed to be fast and efficient, handling large projects with ease. Common operations like committing changes, branching, and merging are quick, even for large codebases.
4. **Staging Area:** Git uses a staging area (or index) where changes can be reviewed and modified before committing them to the repository. This allows for more control over what gets included in a commit.
5. **Commit History:** Each set of changes in Git is stored as a commit, which includes a message describing the changes. This creates a detailed history of the project, making it easier to understand how and why changes were made.
6. **Collaboration:** Git supports collaborative workflows, enabling multiple developers to work on the same project simultaneously. It handles conflicts that arise when changes from different sources need to be integrated.
7. **Open Source:** Git is open-source software, meaning it is free to use and has a large community of contributors and users who continually improve and support it.

Common commands in Git include:

- ``git init``: Initialize a new Git repository.
- ``git clone``: Clone an existing repository.
- ``git add``: Stage changes for the next commit.
- ``git commit``: Commit staged changes to the repository.
- ``git push``: Push commits to a remote repository.
- ``git pull``: Fetch and merge changes from a remote repository.
- ``git branch``: List, create, or delete branches.
- ``git checkout``: Switch between branches or restore working tree files.
- ``git merge``: Merge branches together.

Git is widely used in software development and other fields where version control is crucial. Some of the basic GIT vocabulary is:

- Project = repository (repo)
- Working directory (folder where your main repo is)
- Commit: GIT nothing gets change until you commit. It is like a save.
- Staging: getting the files ready or preparing to commit. It is basically a selection of which modified files are going to be committed.
- Push: the purpose of 'git push' is to upload local repository changes to a remote repository. After all committed changes locally, 'git push' is used to send those commits to a remote repository such as GitHub, GitLab, or Bitbucket.
- Pull: the purpose of 'git pull' is to fetch and integrate changes from a remote repository into the local repository. For example, when you want to update your local repository with changes made by others, you use 'git pull'. This command is combination of two commands 'git fetch' (which download changes) and 'git merge' (which integrates changes into the current branch).

To learn more about Git and GitHub, visit <https://education.github.com/git-cheat-sheet-education.pdf>

IDE Text Editing Software

Even git is run through a command-line on your computer. However, there is another that will make working with the command-line a lot easier by using an IDE text-editor. I strongly recommend you use [Visual Studio Code](#) as your text-editing software. Here are three big reasons:

1. It has emerged as the industry standard in recent years.

2. It is entirely free and compatible with Windows, Mac, and Linux.
3. It significantly simplifies command-line operations.

VS Code includes an integrated command-line, eliminating the need to open a separate terminal on your computer. When you open a folder in VS Code (via the File menu -> Open Folder), the built-in terminal automatically navigates to that folder. This convenience means you won't need to use the `cd` command to change directories.

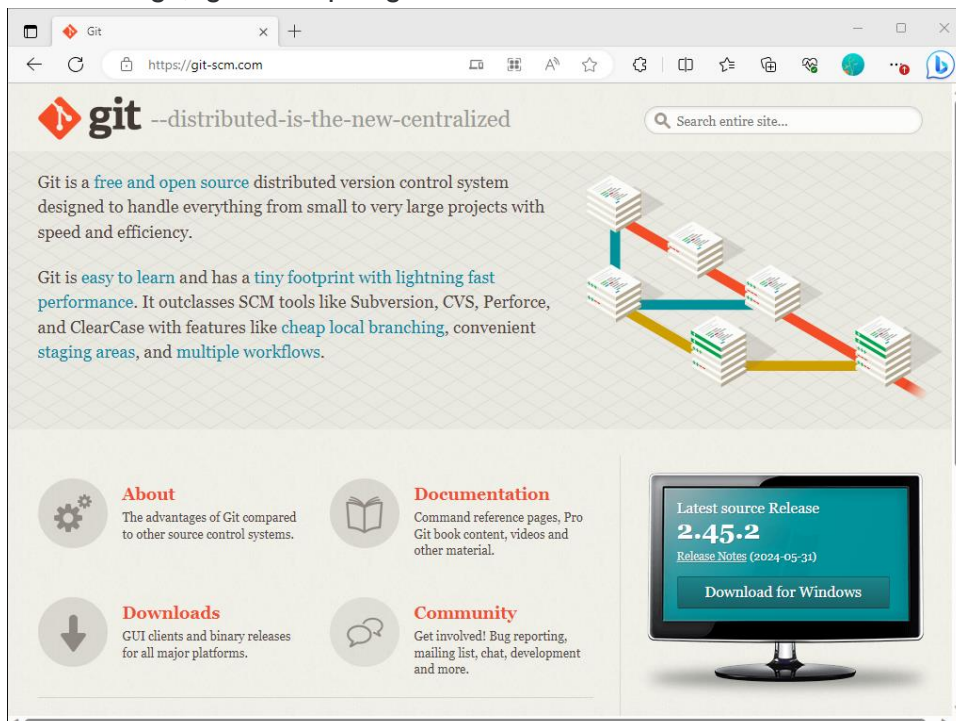
To open the integrated terminal in VS Code, click the View menu and select Terminal. Alternatively, you can use the keyboard shortcut by holding down the control key and pressing the tilde key (located above the tab key).

For Windows users, although you need to download Git, you don't have to use Git Bash as your command-line interface. While the default Windows command-line wasn't previously optimized for web development, the current default PowerShell now performs effectively for these workflows.

Installing git

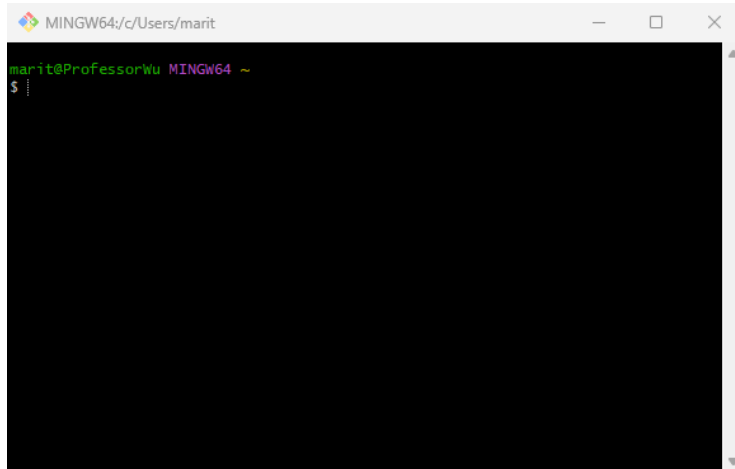
Windows users

To install git, go to <https://git-scm.com/>:



For, Windows users, click on 'Download for Windows' and in the next window, click on 'Click here to download'. After it, run the 'Git.version.exe' file and click 'next' to all options.

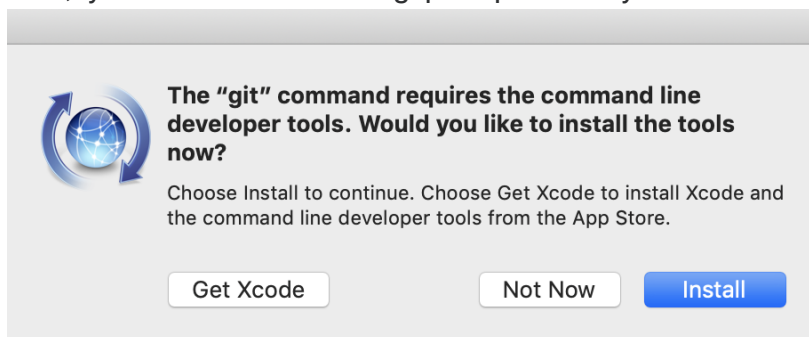
After the installation, it should open the git command prompt as shown below:



If the window does not automatically open, go to 'Search' and look for 'Git Bash'. For more information, you can watch a YouTube video to install git in Windows 11 <https://youtu.be/JNWXQskz4Uk>.

Mac Users

If you are a Mac user, you don't need to install git, git is already part of the Xcode of an OS. To check if Git is already installed in your Mac, open the 'Terminal' and type **git --version**. If Git is installed, you won't need to visit a website to download Git. If Git isn't already installed on your Mac, you'll see the following prompt when you check Git's version:



Instead of visiting the Git website, you can simply click the "Install" button on this Mac prompt, and Git will be installed on your computer. If it is not showing the Windows above, then you can go to <https://git-scm.com/> and download the Mac version.

Setting up your git account

You can personalize a few of your Git settings so when Git tracks your changes, it also tracks who is making those changes. For this, we can set our name and email address by typing:

```
git config --global user.name "professorwu"  
git config --global user.email "hwu@qcc.cuny.edu"
```

For this lab, the 'user.name' should be student's full name and 'user.email' should be student's email

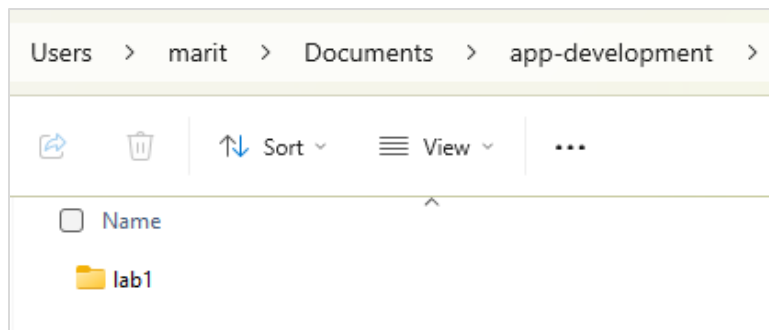
Getting started with Git

To practice some of the Git commands by:

- create a project folder and name it "app-development".
- Check the current directory by typing 'pwd' (print the working directory) from the git Terminal.
- Point the Terminal the project folder named 'app-development' by using the command 'cd' (change directory). To do so, after the command 'cd' you can type out the location of the folder using the forward slash as the folder path indicators. Another way to do so is to just drag the folder 'app-development' into the Git Terminal after the 'cd' command.

For practice, we are going to create a lab folder inside of our 'app-development' directory (folder) by typing the command: `mkdir "lab1"`

you can navigate to the project folder 'app-development' to check if the folder is created:



Now, let us navigate to the 'lab1' folder by typing:
`cd lab1`

Git commands

git init

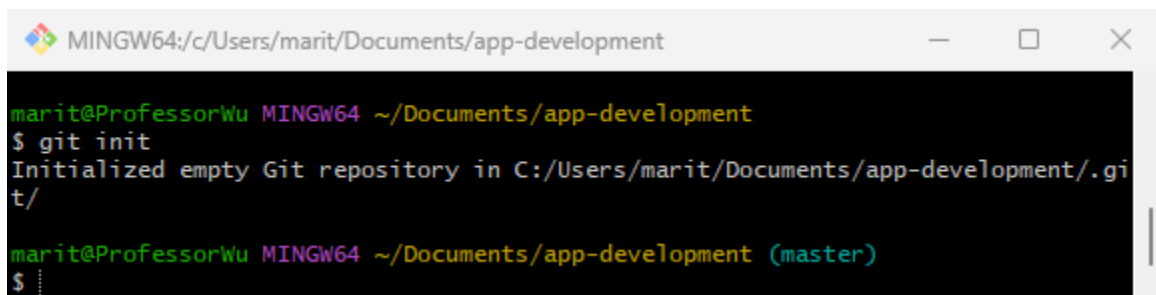
`git init` is a command used to create a new Git repository. It sets up the necessary files and directories for a new Git repository in your current working directory as `.git`. This hidden directory contains all the metadata, including the repository's history, configuration, and current state.

How to Use git init?

From the Git Terminal pointed to 'app-development', type:

```
git init
```

The git Terminal should point to the 'app-development' directory with the word 'master' in parenthesis:

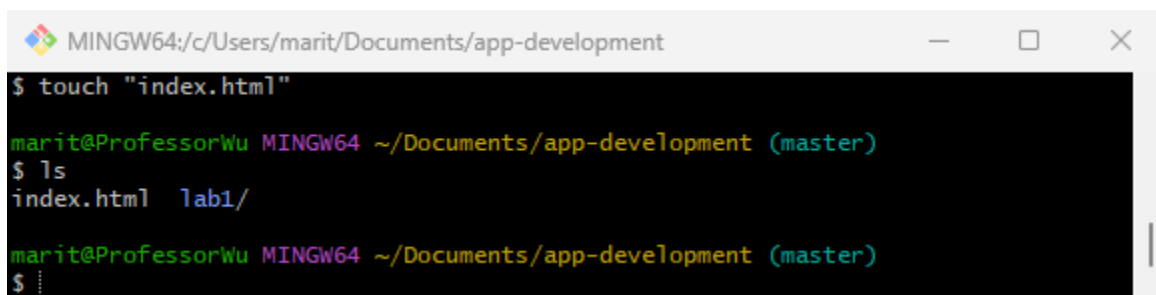
A screenshot of a Windows command prompt window titled "MINGW64:/c/Users/marit/Documents/app-development". The prompt shows the user "marit@ProfessorWu" in the directory "~/Documents/app-development". The command "\$ git init" has been entered, and the output is "Initialized empty Git repository in C:/Users/marit/Documents/app-development/.git/". The prompt now shows "(master)" in parentheses, indicating the current branch.

```
MINGW64:/c/Users/marit/Documents/app-development
marit@ProfessorWu MINGW64 ~/Documents/app-development
$ git init
Initialized empty Git repository in C:/Users/marit/Documents/app-development/.git/
marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$
```

Now, to create a new file, we can use the command 'touch' following by the name of the file. For example, if we want to create the "index.html" file, then we type:

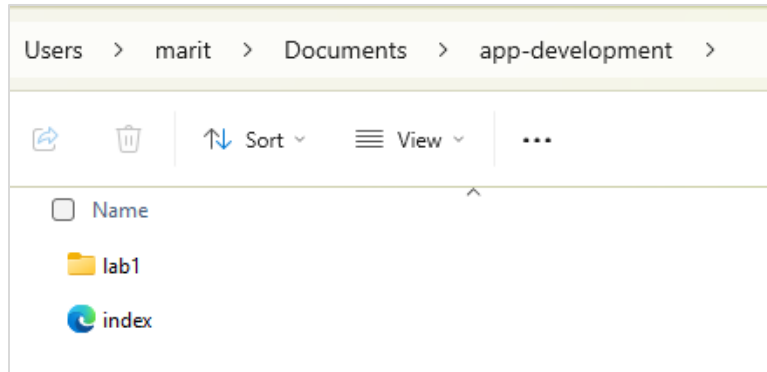
```
touch "index.html"
```

To check, we can type 'ls' (list directory) command to check if the file is created:

A screenshot of a Windows command prompt window titled "MINGW64:/c/Users/marit/Documents/app-development". The prompt shows the user "marit@ProfessorWu" in the directory "~/Documents/app-development (master)". The command "\$ touch \"index.html\"" has been entered. The prompt then shows "\$ ls" and the output "index.html lab1/".

```
MINGW64:/c/Users/marit/Documents/app-development
$ touch "index.html"
marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$ ls
index.html  lab1/
marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$
```

or simply navigate to the directory and check if the file is created:



git status

`git status` is a command used to display the state of the working directory and the staging area in a Git repository. It provides information about which changes have been staged, which haven't, and which files aren't being tracked by Git. This command is very useful to understand the current status of your project before making a commit.

What git status Does

- **Shows Changed Files:** Lists files that have been modified but not yet staged for the next commit.
- **Shows Staged Files:** Lists files that are staged and ready to be committed.
- **Shows Untracked Files:** Lists files that are not being tracked by Git

How to Use 'git status'

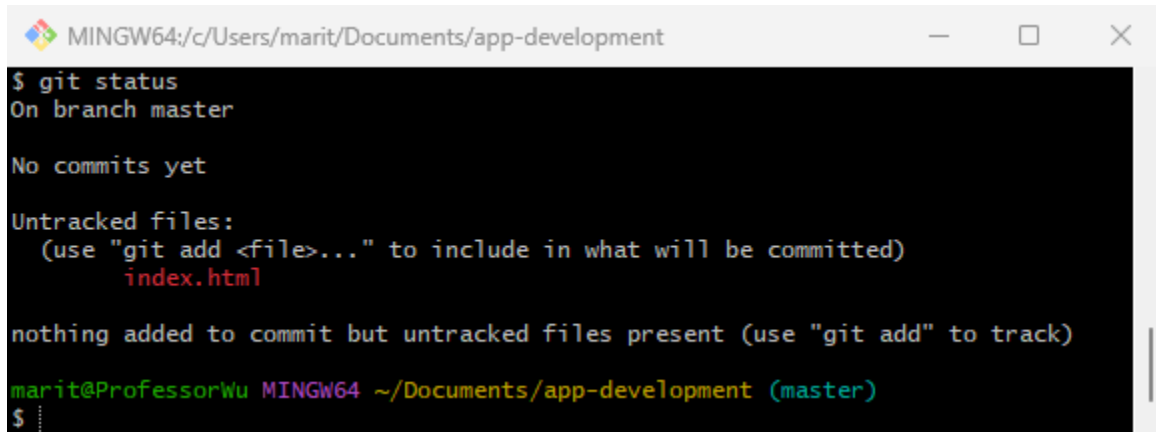
Example) From the project directory, let's open the 'index.html' file in a text-editor, create a HTML5 structure and a `<h1>` tag as:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Welcome to Git</title>
</head>
<body>
  <h1>Student's full name</h1>
</body>
</html>
```

Now, back the git Terminal, type the command:

```
git status
```

The git Terminal will tell us that we have made one file and nothing were added to commit

A screenshot of a terminal window titled 'MINGW64:/c/Users/marit/Documents/app-development'. The terminal shows the output of the 'git status' command. The output indicates that the user is on the 'master' branch, there are no commits yet, and there is one untracked file named 'index.html'. It also provides instructions on how to use 'git add' to track the file. The prompt shows the user is 'marit@ProfessorWu' in the 'MINGW64' environment, located in the directory '~/Documents/app-development' on the 'master' branch.

```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

nothing added to commit but untracked files present (use "git add" to track)

marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$
```

git add

`git add` is a command used to stage changes in your working directory for the next commit. It tells Git to include updates to a particular file or set of files in the next commit. This is an important step in the Git workflow, as it allows you to control which changes you commit to the repository.

What git add Does?

- **Stages Changes:** Adds changes from the working directory to the staging area, preparing them to be committed.
- **Selective Staging:** Allows you to stage specific files or even specific parts of a file.

How to Use git add:

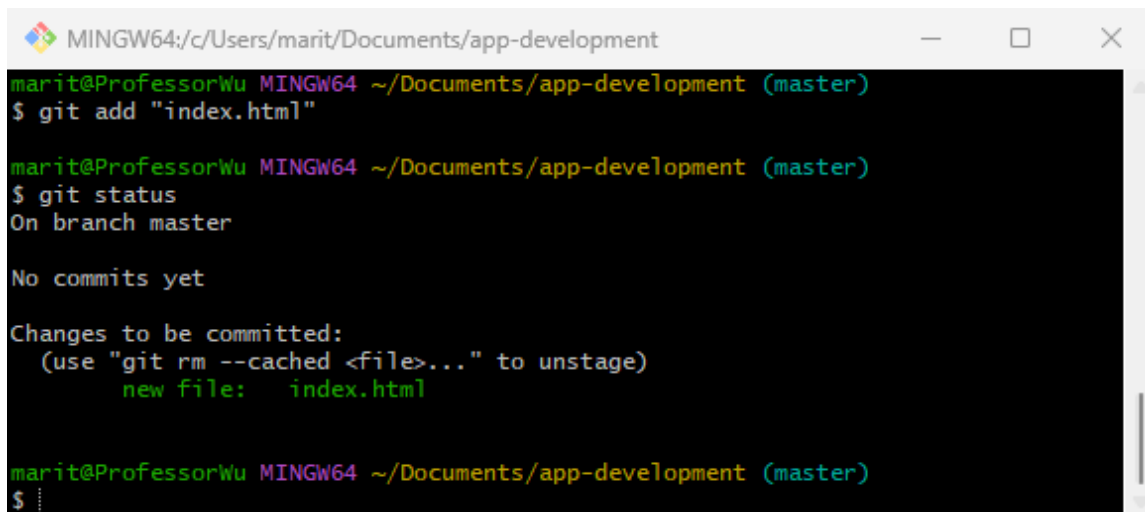
1. **Navigate to Your Project Directory:** for this lab will be our “app-development”
2. **Run git add:** use the command to stage files. There are several ways to use it:
 - Stage a specific file: `git add "filename"`

- Stage Multiple Specific Files: `git add "file1" "file2"`
- Stage All Changes in the Current Directory and Subdirectories. This will stage all modified and new files in the current directory and its subdirectories.: `git add *`
This will stage all modified and new files in the current directory and its subdirectories. This will stage all modified and new files in the current directory only: `git add *`
- Stage a Specific Directory: `git add path/to/directory/`

Example) For our practice, let add the 'index.html' file into a stage by typing:

`git add "index.html"`

Then, we can type `git status` to see if the "index.html" file is added to stage:



```

MINGW64/c/Users/marit/Documents/app-development
marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$ git add "index.html"

marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   index.html

marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$

```

The Terminal, as shown in the image above, it says that file "index.html" is ready to commit.

git commit

`git commit` is a command used to save your changes to the local repository. When you make a commit, you are recording a snapshot of the staged changes along with a message describing what has been done. This is a crucial part of the version control process as it creates a history of changes that can be reviewed, reverted, or referenced later.

What git commit Does?

- **Records Changes:** Captures the current state of the staged changes in the repository.
- **Creates a Commit:** Each commit has a unique identifier (hash), an author, a date, and a commit message describing the changes.
- **Keeps History:** Adds the commit to the project's history, allowing you to track changes over time.

How to Use git commit

1. **Stage Your Changes:** Before committing, ensure that you have staged the changes you want to include in the commit using 'git add'.
2. **Run git commit:** There are several ways to use the git commit command:
 - **Commit with a Message:** The most common way to commit is by providing a message with the **-m** option. For example, `git commit -m "Add new tag <h1> to index.html"`

Best Practices for Commit Messages

- **Be Descriptive:** Clearly describe what changes have been made and why.
 - **Keep it Concise:** Keep the message brief but informative.
 - **Use the Imperative Mood:** Write the message as if you're giving an instruction, e.g., "Add feature" rather than "Added feature".
-
- **commit Without a Message:** If you run git commit without the **-m** option, Git will open a text editor for you to write the commit message.
 - **Amend the Last Commit:** If you need to change the last commit (e.g., to add more changes or correct the message), use the **--amend** option. For example,

```
git add additional_file.txt
git commit --amend
```

This will open the text editor to modify the commit message of the previous commit and include any new staged changes.

3. **View the Commit History:** After making commits, you can view the history of commits using:

```
git log
```

This displays a list of commits with their messages, authors, and dates.

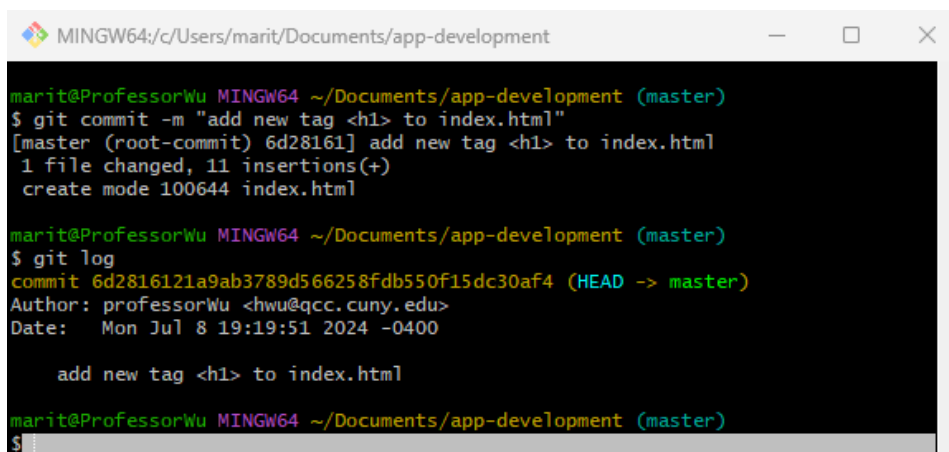
Using git commit effectively helps maintain a clear and organized project history, making it easier to track progress, understand changes, and collaborate with others.

Example) continuing with the previous example, let us commit the “testing-directory” directory with a message as "add new tag <h1>"

```
git commit -m "add new tag <h1> to index.html"
```

we can also check our commit history by typing `git log`

The result of our Terminal will be as follows:

A screenshot of a terminal window titled 'MINGW64; c:/Users/marit/Documents/app-development'. The terminal shows the following commands and output:

```
marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$ git commit -m "add new tag <h1> to index.html"
[master (root-commit) 6d28161] add new tag <h1> to index.html
1 file changed, 11 insertions(+)
create mode 100644 index.html

marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$ git log
commit 6d2816121a9ab3789d566258fdb550f15dc30af4 (HEAD -> master)
Author: professorWu <hwwu@qcc.cuny.edu>
Date: Mon Jul 8 19:19:51 2024 -0400

    add new tag <h1> to index.html

marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$
```

git checkout

`git checkout` is a versatile command in Git used primarily to switch between different branches or to restore files in your working directory to a previous state. Here's how `git checkout` is used in various contexts:

Switching Branches

- **Switch to an Existing Branch:** `git checkout branch_name`

This command switches your working directory to the specified branch (`branch_name`). Any changes that are not committed will be preserved in your current branch or staged area.

- **Create and Switch to a New Branch:** `git checkout -b new_branch_name`

This command creates a new branch (`new_branch_name`) and switches your working directory to this newly created branch.

Restoring Files

- **Restore a Single File to the Last Committed Version:** `git checkout - "file_name"`

This command discards changes in the working directory for `file_name` and replaces it with the version from the last commit.

- **Restore All Files to the Last Committed Version:** `git checkout -- .`

This command discards all changes in the working directory and replaces all files with the versions from the last commit.

Detached HEAD State

- **Inspect a Specific Commit:** `git checkout commit_hash`. This command checks out the specific commit (`commit_hash`) in a detached HEAD state. In this state, you are not on any branch and any new commits will not be part of any branch's history.

For example,

- **Switching Branches:** `git checkout main`. Switches to the `main` branch.
- **Creating and Switching to a New Branch:** `git checkout -b feature_branch`. Creates and switches to a new branch named `feature_branch`.

- **Restoring a File:** `git checkout -- file1.txt`. Discards changes in `file1.txt` and restores it to the last committed version.
- **Detached HEAD State:** `git checkout abc123def456gh789ijk`. Checks out the commit with hash `abc123def456gh789ijk` in a detached HEAD state.

Important Notes

- Always ensure you have committed or stashed any changes before using `git checkout` to switch branches or restore files, as uncommitted changes may be overwritten or lost.
- Use `git checkout` with caution, especially when dealing with the `--` option to restore files, as it permanently replaces changes in your working directory.
- Understanding `git checkout` is fundamental for managing branches, inspecting history, and restoring files to different states in Git.

Example) manually delete the 'index.html' file from the directory folder. Type `git checkout -- "index.html"` to restore index.html file of r `git checkout -- .` to restore all files after the last commit

git clone

`git clone` is a Git command used to create a copy of an existing Git repository. It not only copies the repository itself but also sets up a connection to the remote repository, allowing you to fetch updates and collaborate with others.

What git clone Does

- **Creates a Copy:** Copies an existing Git repository from a remote location (like GitHub, GitLab, or a private server) to your local machine.
- **Sets Up Remote Tracking:** Establishes a connection to the remote repository so you can fetch updates made by others.
- **Preserves History:** Copies all branches, commits, and history from the remote repository to your local machine.

How to Use git clone

- **Clone a Repository:** `git clone <repository_url>`

Replace `<repository_url>` with the URL of the repository you want to clone.

For example: `git clone https://github.com/user/repo.git`

- **Clone into a Specific Directory:** By default, `git clone` creates a new directory with the repository name and clones into it. You can specify a different directory name: `git clone <repository_url> <directory_name>`

Example: `git clone https://github.com/user/repo.git my_project`

Example Workflow

- **Clone a Repository:** `git clone https://github.com/user/repo.git`

This command clones the repo repository from GitHub to your local machine.

- **Navigate into the Cloned Repository:** `cd repo`


Change into the newly created `repo` directory.





- **View Remote Information:** After cloning, you can view the remote repository URL using: `git remote -v`


Example) clone the current repository into a GitHub account by typing: `git clone https://github.com/user/app-development.git`

Where `user` is the GitHub account user name, and `app-development` is the GitHub repository.

Before the clone, you can create the 'app-development' repository in your GitHub account and then clone the local git project using the GitHub repository URL as shows below:

 **app-development** Public


 Pin  Unwatch 1  Fork 0  Star 0



Set up GitHub Copilot

Use GitHub's AI pair programmer to autocomplete suggestions as you code.

[Get started with GitHub Copilot](#)




Add collaborators to this repository

Search for people using their GitHub username or email address.

[Invite collaborators](#)

Quick setup — if you've done this kind of thing before

 Set up in Desktop or HTTPS SSH

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# app-development" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/huixinwu/app-development.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/huixinwu/app-development.git
git branch -M main
git push -u origin main
```

After running the code, 'Connect to GitHub' window will appear:

MINGW64:/c:/Users/marit/Documents/microcredential_Advanced/lab1/testing-directory

```
marit@ProfessorWu MINGW64 ~/Documents/microcredential_Advanced/lab1/testing-directory (master)
$ git clone https://github.com/huixin/app-development.git
Cloning into 'app-development'...
```

Connect to GitHub

GitHub
Sign in

Browser/Device

Token

Sign in with your browser

Sign in with a code

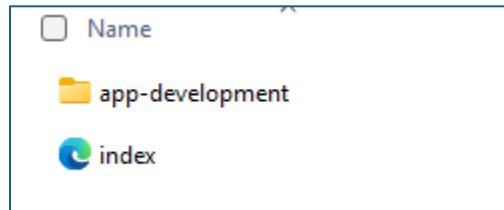
Don't have an account? [Sign up](#)

After you make the connection, run the line again:

```
MINGW64:/c:/Users/marit/Documents/microcredential_Advanced/lab1/testing-directory
marit@ProfessorWu MINGW64 ~/Documents/microcredential_Advanced/lab1/testing-directory (master)
$ git clone https://github.com/huixinwu/app-development.git
Cloning into 'app-development'...
warning: You appear to have cloned an empty repository.

marit@ProfessorWu MINGW64 ~/Documents/microcredential_Advanced/lab1/testing-directory (master)
$
```

Checking the folder:



Host our git repository in GitHub

Hosting a Git repository on GitHub involves a few straightforward steps. Here's a guide to get you started:

Steps to host a Git repository on GitHub

1. **Create a GitHub Account:** If you don't already have one, sign up for a GitHub account at github.com.
2. **Create a New Repository:** Once logged in, click on the "+" sign in the upper right corner of the GitHub homepage and select "New repository".
 - Choose a name for your repository. Optionally, add a description.
 - Decide whether the repository should be public (visible to everyone) or private (accessible only to you and collaborators, requires a paid GitHub plan for private repositories).
3. **Initialize with a README (Optional):**

You can choose to initialize the repository with a README file. This is useful for providing an initial description of your project.
4. **Create Repository:** Click on the "Create repository" button to create your new repository on GitHub.
5. **Set Up Local Repository:**
 - If you haven't already done so, initialize a new Git repository locally or use an existing one.


```
git init
```

- Add your files to the repository and make your initial commit.

```
git add  
git commit -m "Initial commit"
```

6. Link Local Repository to GitHub Repository:

- Link your local repository to the GitHub repository you created using its URL.

```
git remote add origin https://github.com/yourusername/repository.git
```

Replace `yourusername` with your GitHub username and repository with the name of your repository.

7. Push Local Changes to GitHub:

- Push your local repository to GitHub to upload all your committed changes.

```
git push -u origin main
```

This command pushes your changes from the main branch (replace main with the name of your main branch if different).

- **Verify on GitHub:** Refresh your GitHub repository page. You should see your files and commit history now available on GitHub.

Additional Steps

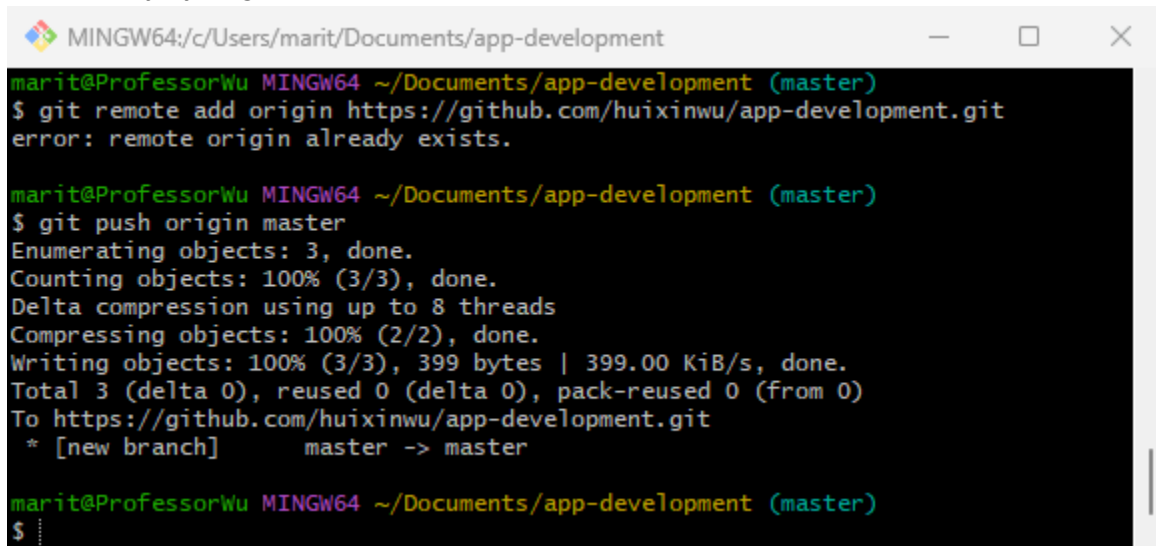
- **Branches and Collaboration:** GitHub supports branching, pull requests, and other collaboration features. Explore these options to manage and collaborate on your project effectively.
- **SSH Authentication:** For easier and more secure access to your GitHub repository, consider setting up SSH keys. GitHub provides detailed instructions for this in their documentation.

Hosting your Git repository on GitHub provides numerous benefits, including easy collaboration, version history tracking, and integration with various development tools and workflows. It's a popular choice for both personal projects and team collaborations.

Example) use the local repository "app-development" and push it to your GitHub repository.

```
git remote add origin https://github.com/huixinwu/app-development.git
```

once the connection is established, we can push our local repository into the GitHub account by typing `git push origin master`

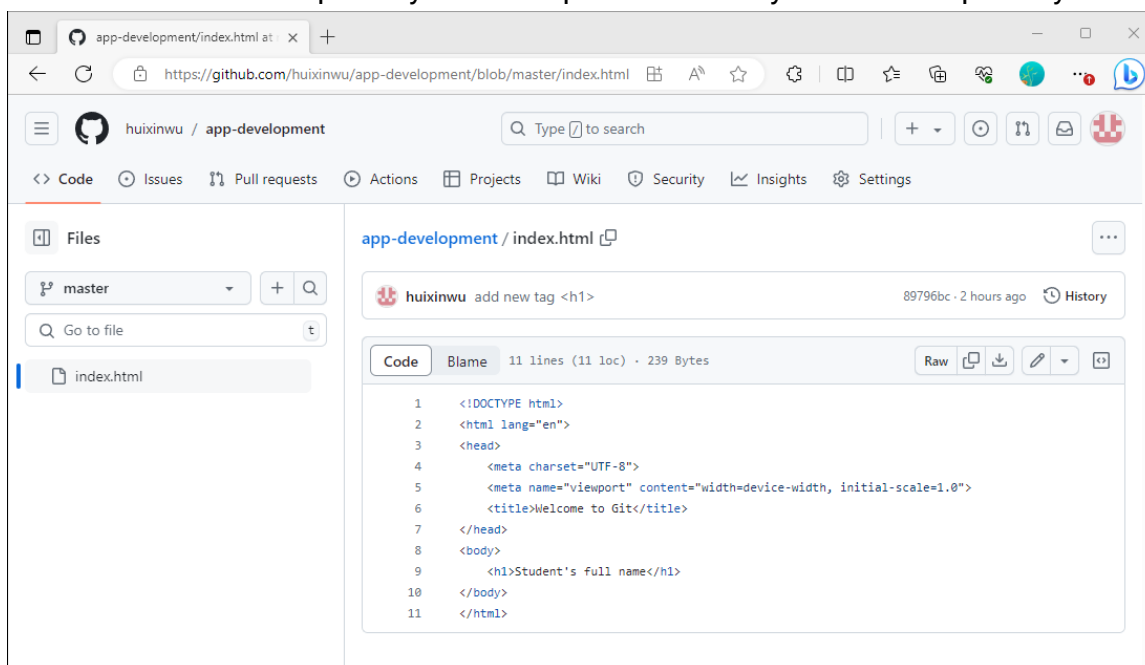


```
MINGW64:/c/Users/marit/Documents/app-development
marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$ git remote add origin https://github.com/huixinwu/app-development.git
error: remote origin already exists.

marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 399 bytes | 399.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/huixinwu/app-development.git
 * [new branch]      master -> master

marit@ProfessorWu MINGW64 ~/Documents/app-development (master)
$
```

You can check the repository for the upload file into your GitHub repository.



git push

`git push` is a Git command used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repository.

What git push Does

- **Uploads Local Changes:** Transfers the committed changes from your local repository to the remote repository.
- **Updates Remote Branches:** Synchronizes the branches in the remote repository with your local branches.

How to Use git push

- **Ensure Your Local Repository is Up to Date:** It's a good practice to pull changes from the remote repository before pushing your changes to avoid conflicts.
`git pull origin main`
Replace main with the name of your branch if it is different.
- **Push Changes to the Remote Repository:**
`git push origin main`
Replace main with the name of your branch if it is different.
- **Push All Branches:** If you want to push all branches to the remote repository, use:
`git push --all origin`
- **Set the Upstream Branch:** When you push a branch for the first time, you might need to set the upstream branch. This tells Git to remember which remote branch to push to.

```
git push -u origin main
```

The -u flag sets the upstream branch, so future pushes can be done with just `git push`.