

## 1. Introduction to Git and GitHub

Git is a **version control system** that helps you keep track of changes in your project files. GitHub is an **online platform** where you can store your Git repositories, collaborate with others, and access your work from anywhere.

For **web programming** in IS424, Git and GitHub are especially useful because they:

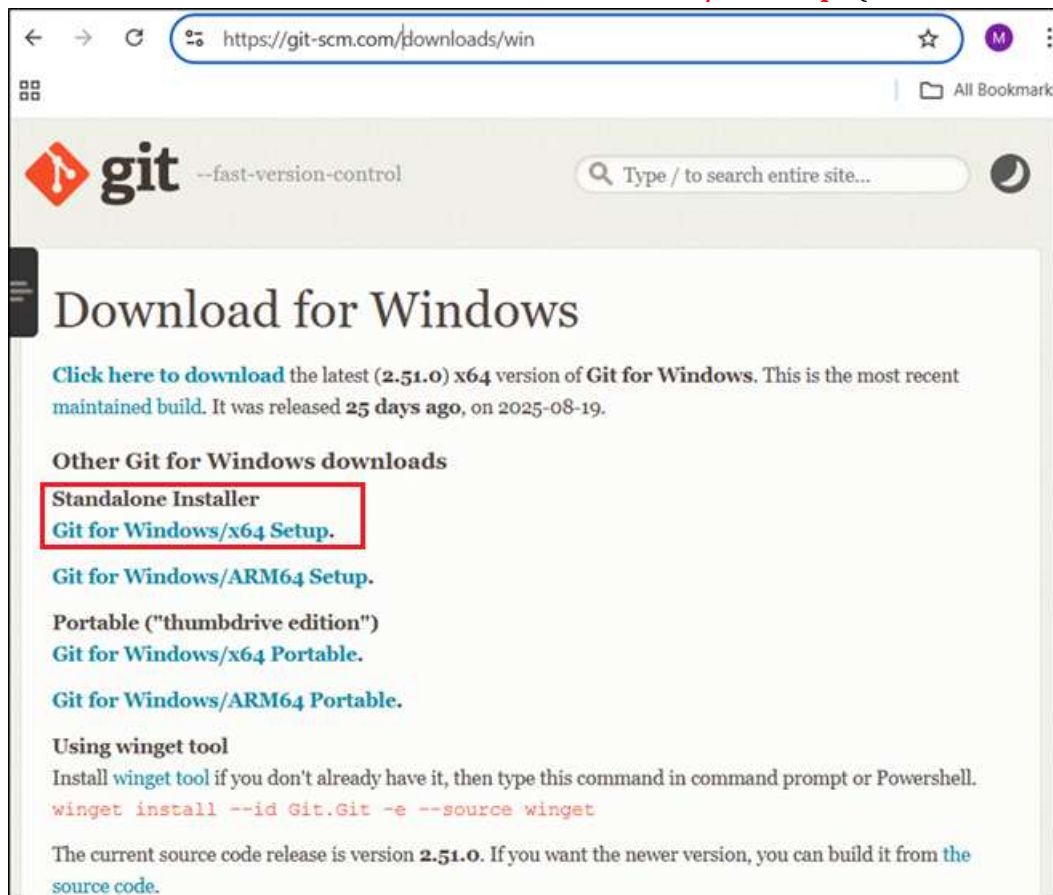
- Keep a **history** of all your code changes
- Allow you to **work safely** without losing previous versions
- Make it easy to **share projects** with instructors and teammates
- Prepare you for **real-world software development practices**

This tutorial introduces Git and GitHub using a single HTML file as a practical example. Students will learn essential version control concepts step by step, starting from basic commands to intermediate topics like branching, merging, and conflict resolution.

## 2. Installing Git

**Windows:** Download Git from <https://git-scm.com/downloads/win> and follow default options.

- Standard installer for **64-bit Windows** “**Git for Windows/64 Setup**” (most modern PCs).



- **Note:** For beginner students, it is safe to keep **all the default options** during the Git installation process — just click **Next** at each step unless instructed otherwise in this tutorial.
  - Default options work well for all course projects (HTML, JavaScript, Python, Anaconda), so students can click **Next** at each screen unless instructed otherwise.

**Mac:** Use 'brew install git' or Xcode Command Line Tools.

**Linux:** Use 'sudo apt-get install git' (Debian/Ubuntu) or 'sudo dnf install git' (Fedora).

#### Check installation with:

```
git --version
```

### 3. First-Time Setup

Run the following commands once after installing Git to set your name and email. This ensures your commits are correctly identified and linked to your GitHub account.

```
git config --global user.name "Your Name"
git config --global user.email your@email.com
```

#### Example:

```
git config --global user.name "Mourad Benchikh"
git config --global user.email "benchikhm@ksu.edu.sa"
```

- **Common Mistake:** Make sure to type **two dashes (--)** in `--global`. Using a single dash (`-global`) will give an error because Git won't recognize the option.

#### Note

This step is **mandatory** because Git will not allow you to commit until at least a username and email are configured, and it ensures your commits are correctly linked to your GitHub account.


### 4. Creating a GitHub Account

- **Go to GitHub**
  - Open your browser and go to <https://github.com>.
- **Sign Up**
  - Click **Sign Up** in the top-right corner (if you have a prior account, then **Sign in**).



- Enter the following information:
  - **Email address**
  - **Password**
  - **Username**
  - **Email preferences** (optional)
  - **And other mandatory information**

## Sign up for GitHub

 Continue with Google

or

Email\*

Email

Password\*

Password

Password should be at least 15 characters OR at least 8 characters including a number and a lowercase letter.

Username\*

Username

Username may only contain alphanumeric characters or single hyphens, and cannot begin or end with a hyphen.

Your Country/Region\*

Saudi Arabia ▼

For compliance reasons, we're required to collect country information to send you occasional updates and announcements.

Email preferences

☐ Receive occasional product updates and announcements

Create account >

### ▪ **Verify Your Email**

- Check your inbox for a verification email from GitHub.
- Click the verification link to activate your account.

### ▪ **Choose a Plan**

- For students, the **Free plan** is enough.

- It allows both **private** and **public** repositories.
- **Optional Profile Setup**
  - Add a profile picture, bio, or other information if you wish.
- **Log In**
  - After logging in, you can create a new.

## 5. Creating a New Repository

### What is a Repository?

A **repository** (or “**repo**”) in Git/GitHub is like a **project folder** that:

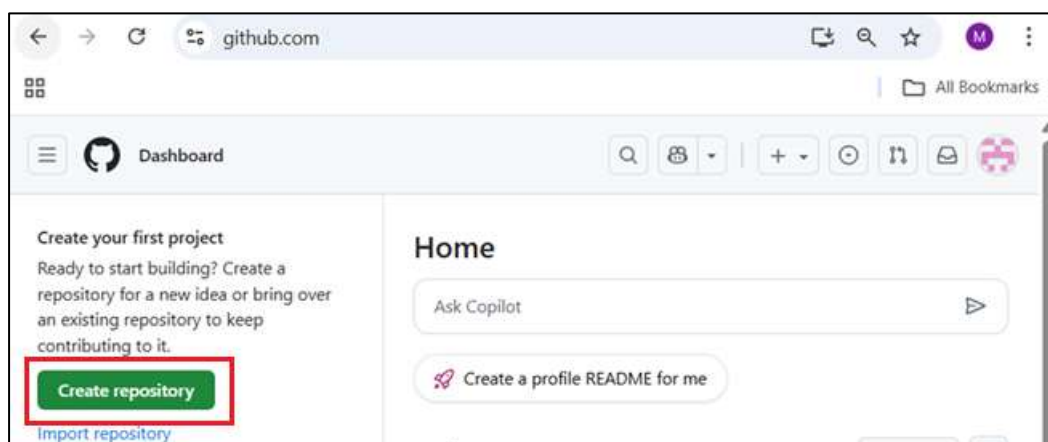
- Stores all your project files (e.g., code, documents, images).
- Keeps a complete **history of changes** made to those files over time.
- Allows you to **synchronize** work between your computer (local repository) and GitHub (remote repository).

In simple terms, a repository is the **central place** where your project lives, both online (on GitHub) and on your computer after you clone it.

We will show next, the steps to create a new repository.

### Step 1: Log in to GitHub

Click the “+” icon in the top-right corner → select “New Repository”.



### Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository](#).  
Required fields are marked with an asterisk (\*).

- #### General

Owner \* benchikhm / Repository name \* IS424  
✓ IS424 is available.

Great repository names are short and memorable. How about **fuzzy-disco**?

Description  
  
0 / 350 characters
- #### Configuration

Choose visibility \*  
Choose who can see and commit to this repository Public

Add README  
READMEs can be used as longer descriptions. [About READMEs](#) Off

Add .gitignore  
.gitignore tells git which files not to track. [About ignoring files](#) No .gitignore

Add license  
Licenses explain how others can use your code. [About licenses](#) No license

[Create repository](#)

## Step 2: Enter Repository Details

- Name your repository: e.g., IS424 (figure above)
- (Optional) Add a short description

## Step 3: Choose Visibility

Use this table to decide whether to make your repository Public or Private:

Option	Who Can See It	Recommended For
Public	Anyone on the internet	Simple class exercises, assignments where links are shared, projects without sensitive data
Private	You + invited collaborators	Projects with private data, large team projects requiring restricted access

For our class, we use **Public** repositories (figure above).

## Step 4: Do Not Initialize with README

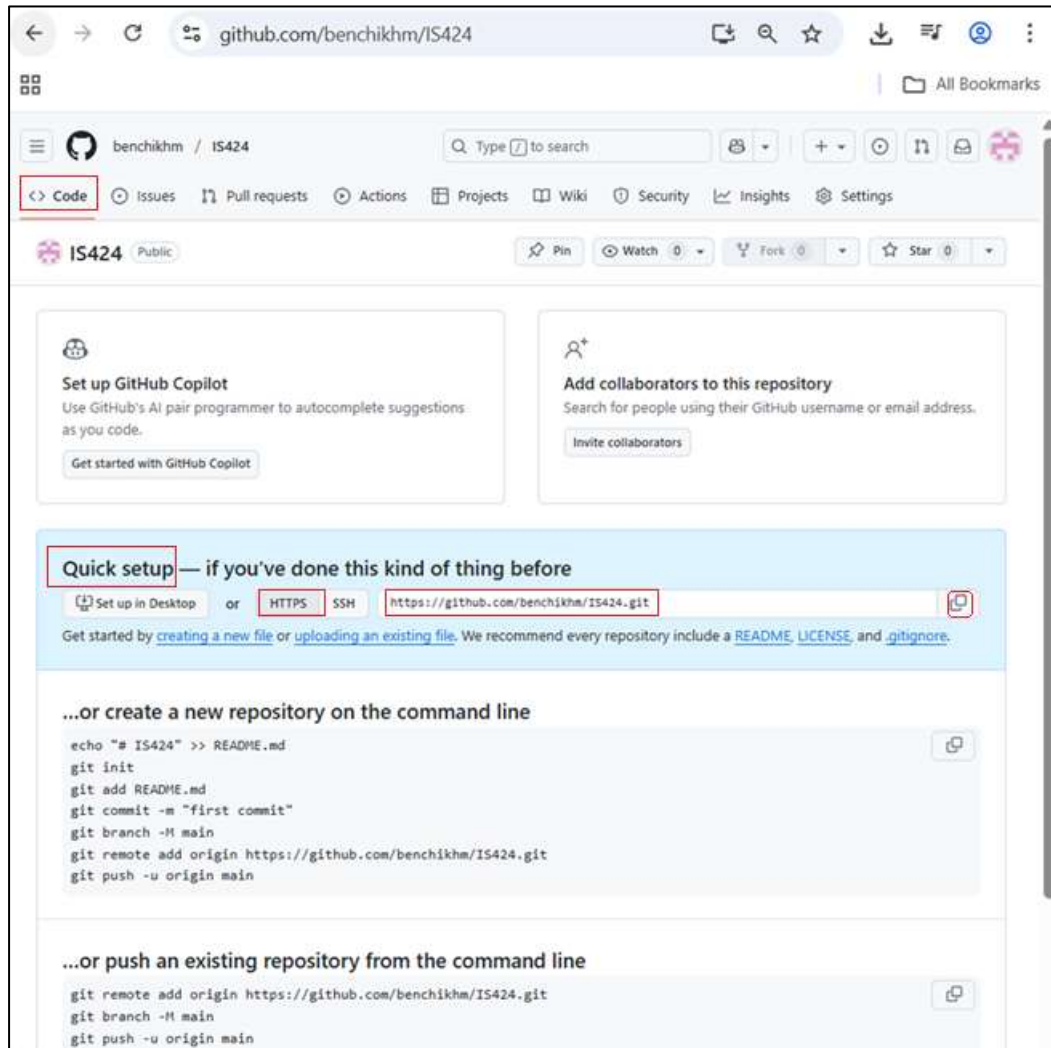
Do NOT check "Initialize this repository with a README." We will add files later from the local machine.

## Step 5: Create the Repository

Click the “Create repository” button (locate it from the above figure).

## Step 6: Copy the Clone URL

After creation, click the Code button.

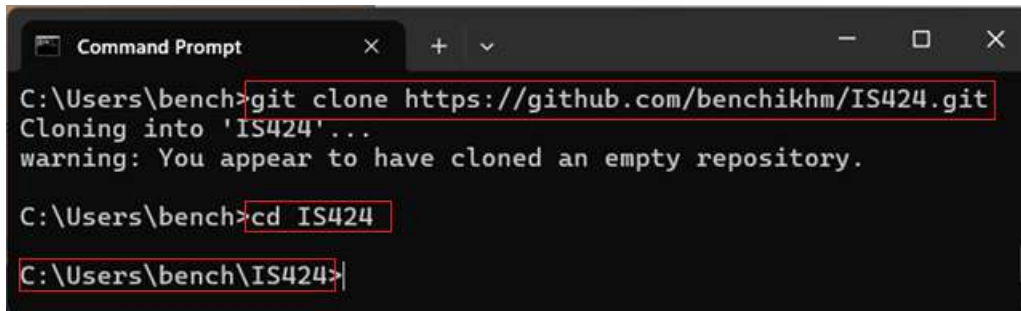


If the repository is empty (**our case**), the same URL appears in the Quick setup box (figure above)— it already ends with .git. Use this URL (by copying it using the Copy button ) with the git clone command:

```
git clone <repository-URL>
cd <repository-name>
```

### Example:

```
git clone https://github.com/benchikhm/IS424.git
cd IS424
```



```
Command Prompt
C:\Users\bench>git clone https://github.com/benchikhm/IS424.git
Cloning into 'IS424'...
warning: You appear to have cloned an empty repository.

C:\Users\bench>cd IS424

C:\Users\bench\IS424>|
```

**Note:** The git clone command creates a folder with the same name as the repository (e.g., IS424) in the directory where you run the command. For example, if you run git clone in C:\Users\Mourad, the folder will be C:\Users\Mourad\IS424.

## OS-Specific Commands for Folder Management and Navigation

This table provides a quick reference for common folder and navigation commands on both Windows and Mac/Linux systems. It can be included in Section #5 of Lab#2 right after the repository creation and cloning steps.

Task	Windows (CMD)	Mac / Linux (Terminal)	Purpose
List files	dir	ls	Show all files in the current folder
Change directory	cd foldername	cd foldername	Move into a folder
Move up one directory	cd ..	cd ..	Move one level up in the folder structure
Create a folder	mkdir foldername	mkdir foldername	Create a new folder
Remove a folder	rmdir /S /Q foldername	rm -rf foldername	Delete a folder and all its contents
Remove a file	del filename	rm filename	Delete a single file

### Step 7: What git clone Does

The git clone command creates a full copy of the GitHub repository on your computer, including all files and version history, and automatically links it to the remote repository. After cloning, you have a local copy connected to GitHub; all commits will stay in sync with this remote repository.

### Step 8: Advanced Users (Optional)

If you start coding locally before creating a GitHub repository, link it manually using:

```
git init
git remote add origin <repository-URL>
git push -u origin main
```

Beginners can skip this — cloning is simpler and preferred in this tutorial.

**Note:** Always choose HTTPS for this tutorial. SSH requires extra setup (keys) and is for advanced users.

## 6. Git Workflow

The Git workflow moves **changes** through **four main stages** in order:



1) Working Directory → 2) Staging Area → 3) Local Repository → 4) Remote Repository (GitHub)

### What “Changes” Mean in Git

In Git, *changes* refer to any **modifications, additions, or deletions** made to files in your repository.

- This includes **programming files** (e.g., .html, .css, .js, .py),
- **documentation files** (e.g., .md, .txt, .docx), and
- **any other files** in the repository.

Git tracks changes based on file content, not file type

Commands like `git add`, `git commit`, `git push`, and `git pull` move changes through these stages in a simple but powerful process:

- `git add` → moves changes from Working Directory → Staging Area
- `git commit` → saves changes from Staging Area → Local Repository
- `git push` → uploads changes from Local Repository → Remote Repository (GitHub)
- `git pull` → brings updates from Remote Repository → Local Repository

### Git Workflow Diagram (conceptual)



This diagram shows the **conceptual data flow** through the four stages:

- Files start in the **Working Directory** where you create or edit them.
- `git add` moves changes to the **Staging Area**.
- `git commit` saves them to the **Local Repository**.
- `git push` sends commits to the **Remote Repository (GitHub)**.
- `git pull` brings updates from GitHub back to your local repository.

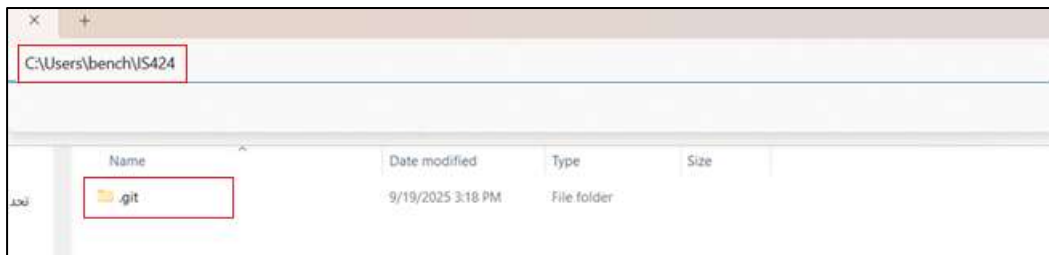
### Folder Structure and Repositories (physical)





This diagram shows the **physical locations**:

- **Working Directory** → The folder on your computer where you edit files (e.g., `C:/Users/bench/IS424`).
- **Local Repository (.git)** → A hidden folder inside the working directory storing Git history and commits (created automatically when you run `git init` or `git clone`).



- **Remote Repository (GitHub)** → The online version of your project for collaboration and backup.

Arrows show how changes move:

- `git add` & `git commit` → Save changes locally.
- `git push` → Send commits to GitHub.
- `git pull` → Get updates from GitHub to your computer.

#### Note:

The **Local Repository** is the hidden `.git` folder inside your project directory. It stores both the **staging area (Index)** and the **commit history**. We show the **Staging Area** separately in the conceptual diagram to illustrate the workflow steps (`git add` → `git commit`), but physically it lives inside the `.git` folder.

## Detailed Explanation of Each Stage

### 1. Working Directory

- This is your local folder on your computer where you create or edit files before Git starts tracking them.
- Example: Editing `index.html` in VS Code.

### 2. Staging Area (Index) (منطقة تحضير)

- Files you want to include in the next commit are staged here using:
  - `git add <file>`
- Think of it as a “**to-do list**” of changes you want to save.
- **Clarification:**
  - When we say *changes*, we do **not** mean the physical files themselves. Your files always stay in the working directory on your computer. What Git records in the **Staging Area** are the *current states* (snapshots (لقطات)) of those files — basically what has changed since the last commit. Later, `git commit` saves these staged snapshots into the local repository’s history, but the files themselves never move out of your working directory.

### 3. Local Repository

- A database on your computer where Git saves snapshots (commits) of your staged changes.
- Use:
  - `git commit -m "message"`
- Commits stay only on your computer until you push them to GitHub.

- **Note:** In Git, a *snapshot* is a **saved version of your entire project** at a specific point in time. Every time you run `git commit`, Git stores one snapshot in the **local repository** so you can view, compare, or restore it later. A snapshot does **not** mean moving your actual files; they stay in your working directory while Git keeps track of their history inside the `.git` folder.

#### 4. Remote Repository (GitHub)

- The version of your repository stored online on GitHub.
- Share your commits with:
  - `git push`
- Get updates from others with:
  - `git pull`

## Key Takeaways

- **Working Directory** → where you edit files.
- **Staging Area** → files selected to be saved.
- **Local Repository** → where snapshots (commits) are stored on your computer.
- **Remote Repository** → GitHub version for sharing and collaboration.

## 7. Core Commands

The following commands are used in order as you make changes to your project. Each command will be demonstrated step-by-step in the next section when editing the `index.html` file in your `IS424` repository.

**Note:** Commands like `ls` (list files) or `touch` (create a new file) are **optional** for this lab. Students can use **File Explorer** or **VS Code** to view files or create new ones instead of typing these OS commands in the terminal.

### 1. Check Status

- `git status`
  - **Example:** `C:\Users\bench\IS424> git status`

Shows the current state of the repository: which files are modified, staged, or committed.

**Note:** Use this often before running the next commands to avoid mistakes.

### 2. Add File to Staging Area

- `git add index.html`
  - **Example:** `C:\Users\bench\IS424> git add index.html`

Prepares the file for committing.

**Note:** This stages only `index.html`. Use `git add .` to stage all modified files at once if you want them in the same commit.

### 3. Commit Changes Locally

- `git commit -m "Describe your change here"`
  - **Example:** `C:\Users\bench\IS424> git commit -m "Add heading to index.html"`

Saves the staged changes to the local repository with a descriptive message.

**Note:** Each commit saves a *snapshot* of all staged changes. Use clear commit messages to document your work properly.

#### 4. Push Changes to GitHub

- `git push -u origin main`
  - **Example:** `C:\Users\bench\IS424> git push -u origin main`

Uploads the committed changes from your local repository to the remote GitHub repository.

**Note:**

- `origin` = the default name of your remote repository
- `main` = the default main branch name in modern Git

#### 5. Pull Updates from GitHub

- `git pull`
  - **Example:** `C:\Users\bench\IS424> git pull`

Fetches and merges updates from GitHub into your local repository.

**Note:** `git pull` combines two steps:

- `git fetch` (downloads updates)
- `git merge` (applies them locally)

#### 6. View Commit History

- `git log --oneline`
  - **Example:** `C:\Users\bench\IS424> git log --oneline`

Shows a simple history of commits for quick reference.

**Note:** Use `git log` without `--oneline` for full commit details.

### 7.1 Removing a Repository

This section shows how to remove a repository both online (on GitHub) and locally (on your computer). Students can use this if they no longer need the repository or want to permanently delete it.

#### Steps to Remove Repository on GitHub (Online):

1. Go to your repository page on GitHub.
2. Click Settings → Danger Zone (bottom of the page).
3. Click Delete this repository.
4. Type the repository name to confirm deletion.

Effect: This permanently deletes all code, branches, commits, and issues from GitHub.

#### Steps to Remove Repository Locally (Windows):

On your Command Prompt (CMD):

```
cd ..  
rmdir /S /Q IS424
```

- `cd ..` → Moves one folder up (so you're outside the repository folder).
- `rmdir /S /Q IS424` → Permanently deletes the folder IS424 and its Git history.

On Windows, you can also delete the folder manually from File Explorer.

## 7.2 Resetting a Repository

Resetting clears all files and commit history so the repository looks like it was just created. This is better than deleting because it keeps the same GitHub link and settings.

### Reset Local Repository Only:

Use this to clear commit history locally while keeping the repository name:

```
git checkout --orphan newbranch  
git add .  
git commit -m "Fresh start"  
git branch -M main  
git push --force origin main
```

Effect: All previous commits are removed, but the repo remains usable with the same name.

### Reset Both Local and Remote Repository:

If you want both local and GitHub repository completely empty:

```
del /Q *.*  
git add .  
git commit -m "Reset repository"  
git push --force origin main
```

Effect: Deletes all files and history from both local and GitHub repositories.

### Recommendation:

Resetting is recommended if students want to keep the repository name and settings but start from scratch. Removing is only necessary if they want to permanently delete the repository.

## 8. Editing index.html Step by Step (Windows Version)

In this section, we will practice the Git commands from Core Commands (#7) using a simple HTML file `index.html`. Each time we make a change, we will:

1. Run the appropriate Git commands,
2. View the output,
3. Push changes to GitHub,
4. Check the results online.

Important: Make sure you are inside your cloned repository folder IS424 before starting:

```
cd IS424
```

## Step 1: Create and Add Initial File

1. Create a new file named `index.html` inside the working directory `IS424`. Example content:

```
<h1>Hello, Git!</h1>
```

2. Save the file.

3. Run the following commands:

**Note:** git is case sensitive, so when dealing with filenames, be aware of that

### git status

```
C:\Users\bench\IS424>git status
On branch main
Your branch is up to date with 'origin/main'.

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)
```

- **On branch main** : You are currently working on the `main` branch.
- **No commits yet** : This is a brand-new repository; no snapshots (commits) exist yet.
- **Untracked files** : `index.html` exists in your working directory but Git isn't tracking changes to it yet.
  - An **untracked file** in this context means a file that exists in your project folder but hasn't been added to Git yet, so Git isn't monitoring its changes.
- **Instruction** : Git even tells you the next step: run `git add index.html` to stage it for the first commit.

### git add index.html

- If you try again "git -status" you will get this

```
C:\Users\bench\IS424>git add Index.html

C:\Users\bench\IS424>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Index.html
```

- Before git add, `index.html` was listed as **Untracked**. Now and after running git add, the file moved to *Changes to be committed*, which mean it is now **staged** in the **Staging Area** (Index).

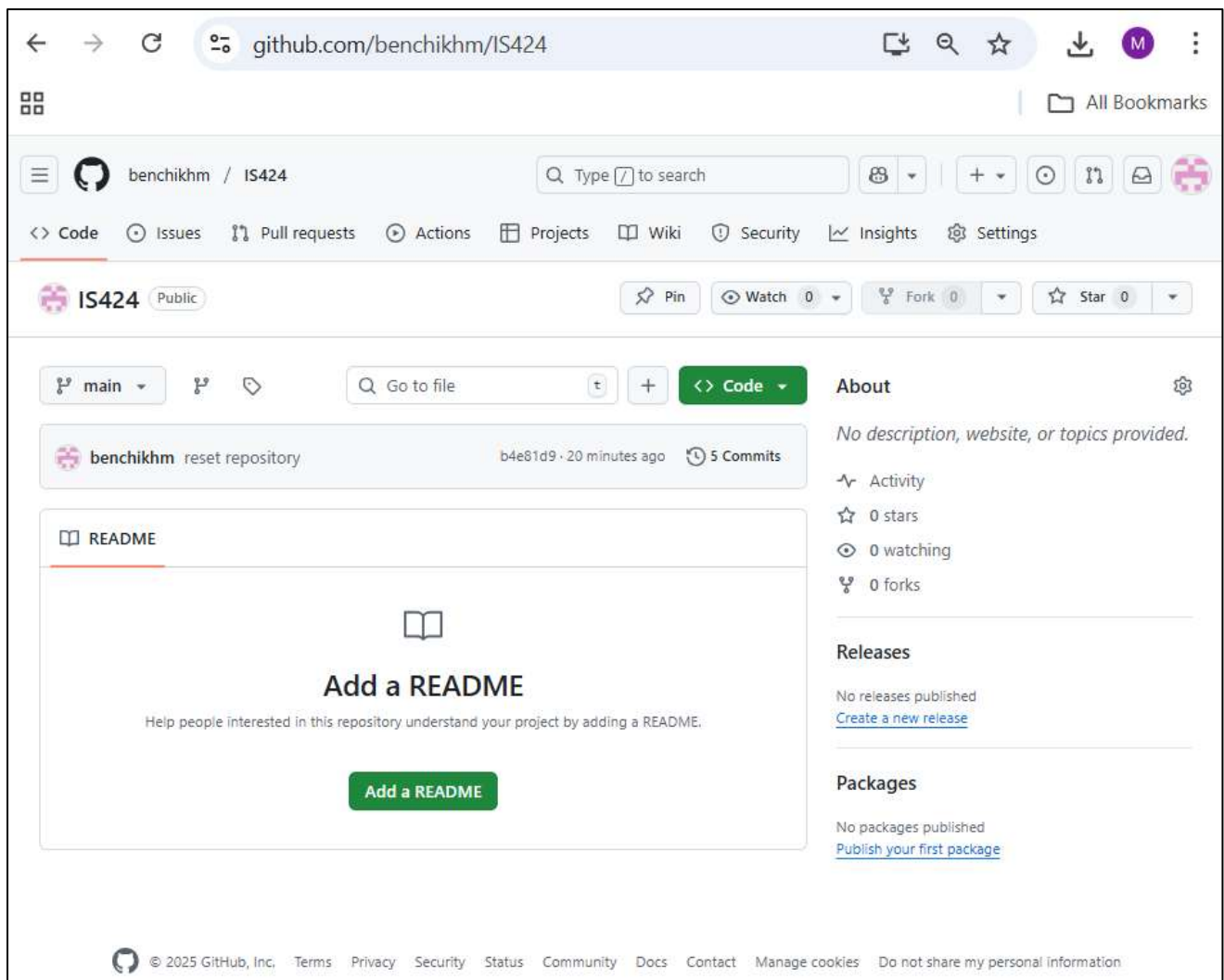
### git commit -m "Add initial index.html"

```
C:\Users\bench\IS424>git commit -m "Add initial Index.html"
[main b5408a1] Add initial Index.html
1 file changed, 1 insertion(+)
create mode 100644 Index.html

C:\Users\bench\IS424>git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
```

- **git commit -m "Add initial Index.html"**
  - **[main b5408a1] Add initial Index.html**
    - Confirms the commit was created on branch **main** with the message *Add initial Index.html*.
  - **1 file changed, 1 insertion(+)**
    - Shows that 1 file (*Index.html*) was added with one line of content (insertion).
  - **create mode 100644 Index.html**
    - This is the file permission (standard read/write for owner, read-only for others).
  - At this point, the **snapshot** of *Index.html* is permanently stored in the **local repository** (*.git* folder).
- **git status**
  - **nothing to commit, working tree clean**
    - Means there are **no pending changes**:
      - The working directory, staging area, and local repository are now synchronized.
      - All changes have been committed successfully.
- We can check that online repository is still empty



## git push -u origin main

```
C:\Users\bench\IS424>git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Writing objects: 100% (3/3), 269 bytes | 269.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/benchikhm/IS424.git
   b4e81d9..b5408a1  main -> main
branch 'main' set up to track 'origin/main'.

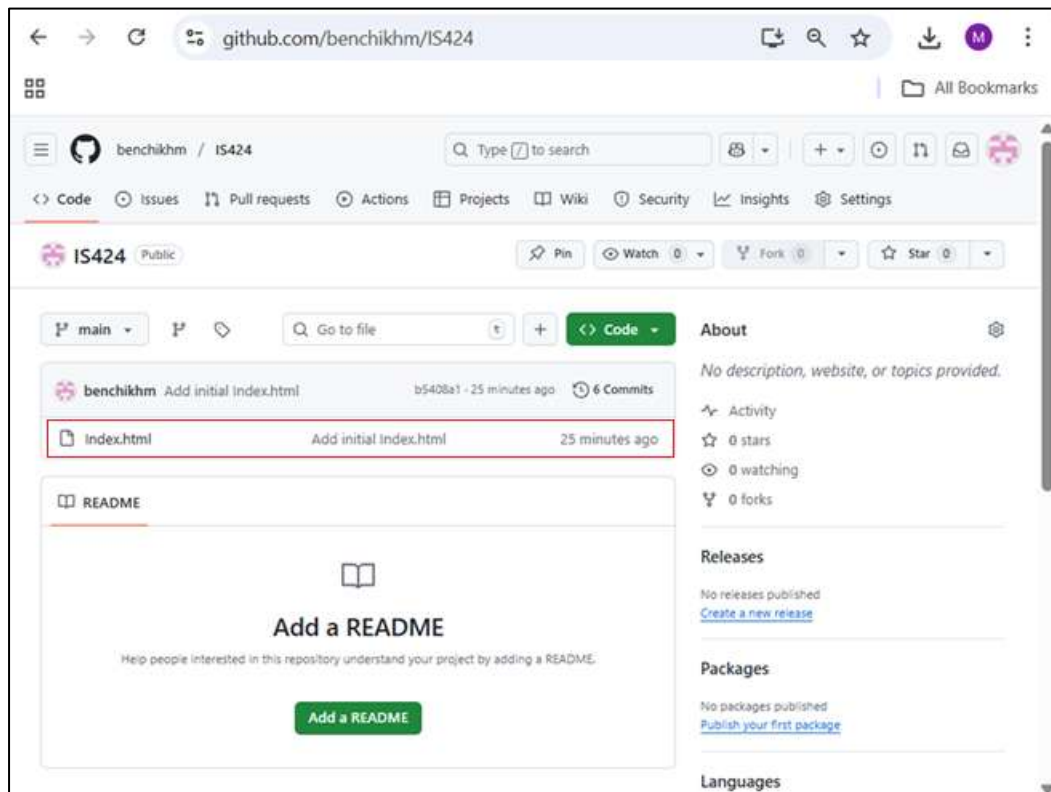
C:\Users\bench\IS424>git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

- **git push -u origin main**
  - This command uploads the committed changes from your **local repository** to the **remote repository** on GitHub (**origin**).
  - The **-u** option sets the **upstream** so that next time you can simply use `git push` without specifying the branch
- **git status after push:**
  - **"Your branch is up to date with 'origin/main'"** → Confirms that **all changes on the local branch** are now synchronized with the **remote branch**.



- “**nothing to commit, working tree clean**” → No pending changes in the working directory or staging area.
- If you refresh your GitHub webpage, you will notice that index.html has indeed been pushed from the local repository to the online (GitHub) repository.



## git log --oneline

- The output shows a **condensed history of commits** in your repository. Here is what each part means:

```

Command Prompt
C:\Users\bench\IS424>git log --oneline
fbc1372 (HEAD -> main, origin/main) Add initial index.html
C:\Users\bench\IS424>

```

- **fbc1372 (HEAD → main, origin/main, origin/HEAD) Add initial Index.html**
  - **fbc1372**: This is the short version of the commit hash (unique ID) for this commit.
  - **HEAD → main**: **HEAD** is a pointer that always refers to your **current commit**. Here, it points to the latest commit on the **main** branch.

- **origin/main, origin/HEAD:** These show that the remote repository on GitHub (called *origin*) is also at the same commit, so your local and remote repositories are fully synchronized.
- **Add initial Index.html:** The commit message you wrote when saving this commit.

#### Explanation:

- **git status:** shows the new file as untracked.
- **git add index.html:** stages the file for commit.
- **git commit:** saves the staged file as a snapshot.
- **git push:** sends the snapshot to GitHub.

## Step 2: Modify File and Commit Changes

1. Edit index.html to add a paragraph:

```
<h1>Hello, Git!</h1>
<p>Learning Git step by step.</p>
```

2. Run:

#### git status

- **On branch main / up to date with 'origin/main':** You are working on the *main* branch, and your local branch is in sync with GitHub.
  - *origin/main* refers to the **main branch on the remote GitHub repository** called *origin*, while *main* alone is your local branch.
- **Changes not staged for commit:** Git detected modifications to *Index.html* since the last commit, but they are **not yet staged** for the next commit.
- **modified: Index.html:** Shows the file with changes in the *working directory*.
- **no changes added to commit:** You need to run `git add Index.html` before you can commit the changes.
- **git restore <file>** suggestion: Git offers this option in case you want to **discard** the changes and go back to the previous state.
  - Git will throw away your paragraph (you added recently) and **restore the file exactly as it was in the last commit** (the first commit in your case).

```
Command Prompt
C:\Users\bench\IS424>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Index.html

no changes added to commit (use "git add" and/or "git commit -a")
C:\Users\bench\IS424>
```

### git add index.html

- Here is what status shows after the command add

```
Command Prompt
C:\Users\bench\IS424>git add Index.html
C:\Users\bench\IS424>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Index.html

C:\Users\bench\IS424>
```

### git commit -m "Add paragraph"

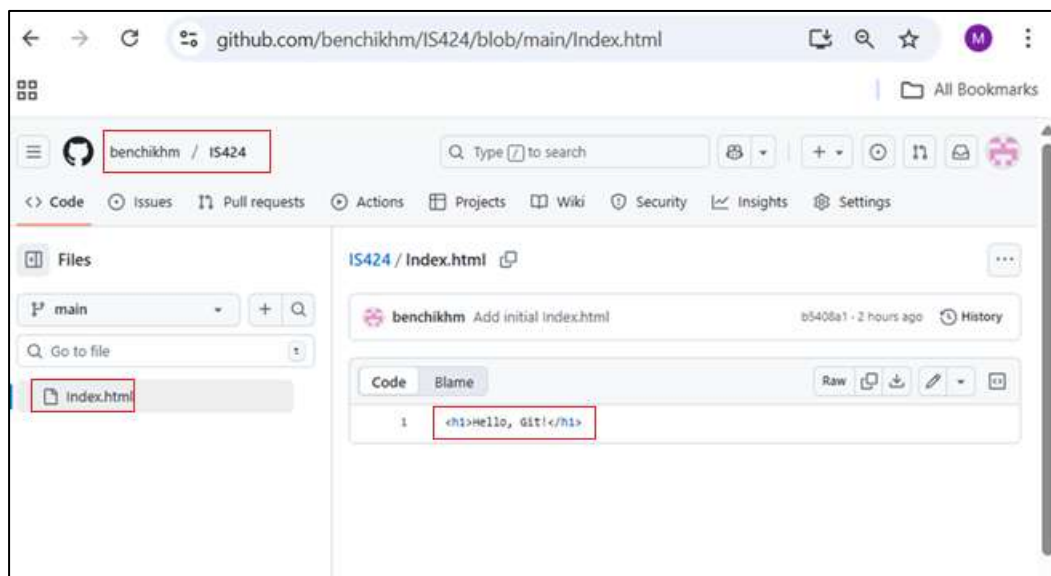
- **git commit -m "Add paragraph"**
  - This command created a new commit with your changes (2 insertions, 1 deletion) and saved it in the **local repository** (.git folder).
  - The unique commit ID is f55f2cd.
- **git status : "Your branch is ahead of 'origin/main' by 1 commit."**
  - This means your **local branch** now has 1 commit that hasn't been pushed yet to the **remote repository** (GitHub).
  - To sync the remote branch with this latest commit, you need to: git push
  - **"nothing to commit, working tree clean"**
    - This tells you there are **no further changes** in your working directory or staging area; everything is already committed locally

```
Command Prompt
C:\Users\bench\IS424>git commit -m "Add paragraph"
[main f55f2cd] Add paragraph
1 file changed, 2 insertions(+), 1 deletion(-)

C:\Users\bench\IS424>git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
C:\Users\bench\IS424>
```

- Here is the remote repository showing that the paragraph is not yet published

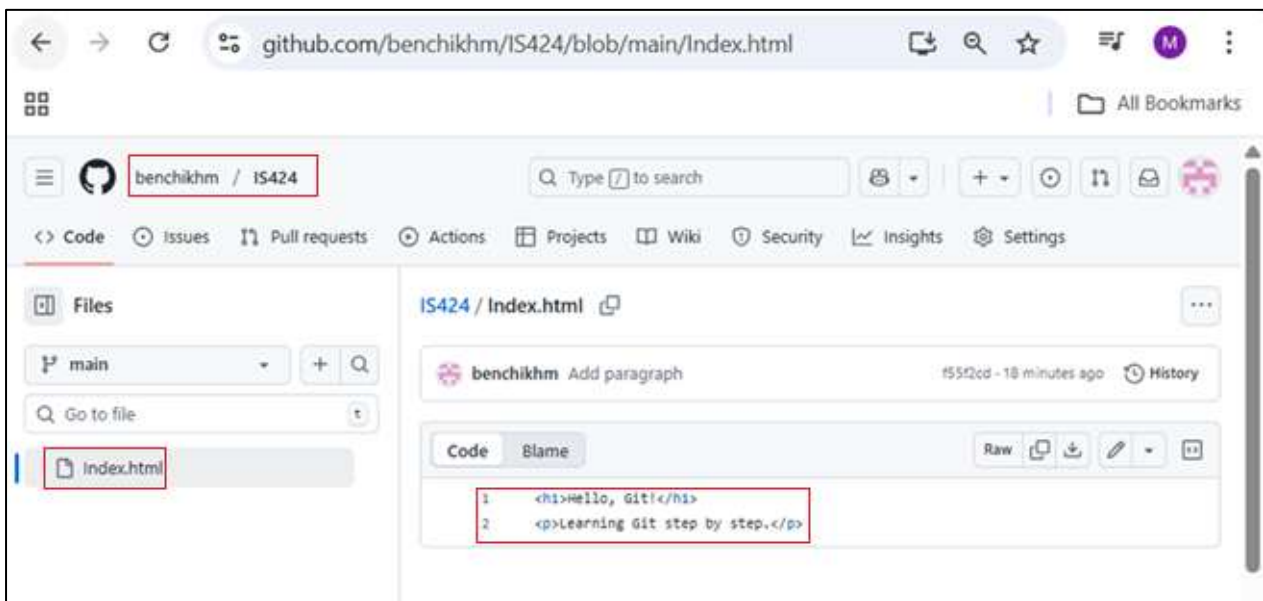


### git push gggit

- **Enumerating objects / Counting objects / Compressing objects / Writing objects:** Git is preparing and sending your changes from the local repository to the remote (GitHub).
- **100% (5/5):** Confirms all objects (changes) have been successfully processed.
- **To <https://github.com/benchikhm/IS424.git>** : Indicates the remote repository URL where changes are pushed.
- **b5408a1..f55f2cd main → main:** Shows that the previous commit (b5408a1) was updated to the latest commit (f55f2cd) on the branch main both locally and remotely.

```
Command Prompt
C:\Users\bench\IS424>git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 290 bytes | 145.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/benchikhm/IS424.git
b5408a1..f55f2cd main -> main
C:\Users\bench\IS424>
```

After this push, the remote GitHub repository now contains the latest changes (the added paragraph).



### git log -online

- **40e0bc8 (HEAD → main, origin/main, origin/HEAD) Add paragraph**
  - **40e0bc8**: Short hash identifying the latest commit.
  - **HEAD → main**: Your local branch `main` points to this commit, so it's the most recent.
  - **origin/main, origin/HEAD**: The remote repository (GitHub) is synchronized at the same commit.
  - **Add paragraph**: The commit message you wrote for the paragraph addition.
- **fbc1372 Add initial Index.html**
  - This was your earlier commit when you first added the `Index.html` file.

```
Command Prompt
C:\Users\bench\IS424>git log --oneline
40e0bc8 (HEAD -> main, origin/main) Add paragraph
fbc1372 Add initial index.html
C:\Users\bench\IS424>
```

**Why run git add again?** Each time you modify the file, Git needs you to stage the new version for the next commit.

### Step 3: Create a Branch for a Feature

What is a branch?

- A branch is like a separate workspace in your repository.
- You can work on new features without affecting the main branch (main).
- When ready, you merge the branch back into main.

1. Create and switch to a new branch:

**git checkout -b feature-footer**

2. Edit index.html to add a footer:

```
<h1>Hello, Git!</h1>
<p>Learning Git step by step.</p>
<footer>End of tutorial example</footer>
```

3. Run:

**git add index.html**

**git commit -m "Add footer in feature branch"**

**git push -u origin feature-footer**

#### Step 4: Merge the Feature Branch into Main

1. Switch back to main branch:

```
git checkout main
```

2. Merge changes:

```
git merge feature-footer
```

3. Push to GitHub:

```
git push
```

#### Step 5: Pull Updates (if needed)

If changes were made on GitHub or by others, bring them to your local machine:

```
git pull
```

- **You should provide a link to the files as an answer in LMS:** "Changes made directly on GitHub can be brought into your local repository using `git pull`, ensuring both local and remote versions stay synchronized."

#### Step 6: Delete Feature Branch (Optional)

After merging, you can delete the feature branch locally:

```
git branch -d feature-footer
```

And on GitHub:

```
git push origin --delete feature-footer
```



## Summary Table

Step	Command Example	Purpose
-----	-----	-----
Create File	git add index.html	Stage new file for commit
Commit	git commit -m "message"	Save snapshot locally
Push	git push	Send commits to GitHub
Branch	git checkout -b feature-footer	Create and switch to feature branch
Merge	git merge feature-footer	Merge feature branch into main
Pull	git pull	Get updates from GitHub
Delete Branch	git branch -d feature-footer	Remove feature branch after merging

## Steps to Provide a Link to the Files in LMS

1. **Go to your GitHub repository**
  - Example: <https://github.com/benchikhm/IS424>
2. **Open the specific file** you want to share.
  - Navigate inside the repository → Click on the file (e.g., `index.html`).
3. **Copy the URL from the browser's address bar**
  - This link will directly open the file in GitHub for anyone with access to your repository.
  - Example: <https://github.com/benchikhm/IS424/blob/main/index.html>
4. **Paste this link as your answer in LMS**
  - If the assignment expects multiple files, you can:
    - Share the **repository link** if all files are there.
    - Or share **individual file links** if needed.
5. **Make sure your repository visibility is correct**
  - If it is **public**, anyone can see it.
  - If it is **private**, you must invite your instructor or collaborators.

## 9. Branching & Merging

Branches allow you to develop features separately without affecting the main code until you merge them

- **git branch** → List branches

- **git checkout -b feature** → Create new branch

- **git checkout main** → Switch back

- **git merge feature** → Merge feature branch into main

After merging, delete the feature branch if it's no longer needed: `git branch -d feature`.

## 10. Handling Merge Conflicts

A merge conflict occurs when two branches change the same line. Git marks conflicts in the file. Open the file, choose the correct version, then:

```
git add index.html  
git commit -m "Resolve merge conflict"
```

Conflicts happen when the same part of a file is changed on two branches. Git marks the conflict using <<<<<<, =====, and >>>>>> lines. You must open the file, choose which changes to keep, delete the markers, then stage and commit the resolved version. Conflicts are marked with

```
<<<<<< HEAD  
...your changes...  
=====  
...incoming changes...  
>>>>>> branch-name
```

## 11. Reset vs Revert

Use git log to find commit hashes.

- **git reset --hard <commit>**: Reset to an old commit (deletes newer commits)

- **git revert <commit>**: Create a new commit that undoes changes safely

Warning: `git reset --hard` deletes commits permanently. Beginners should prefer `git revert`, which creates a new commit to undo changes safely."

## 12. GitHub Extras

1) Forking & Pull Requests: For contributing to other repositories. Pull Requests allow others to review your code before merging it into the main branch

2) GitHub: GitHub Pages: Publish static websites from a repository. Learn more: <https://pages.github.com>

### 13. Git Command Cheat Sheet

git clone <URL> → Copy a remote repo locally  
git status → See what changed  
git add <file> → Stage changes  
git commit -m "msg" → Save snapshot locally  
git push → Send changes to GitHub  
git pull → Get updates from GitHub  
git checkout -b <branch> → Create & switch branch  
git branch → List branches  
git merge <branch> → Merge branch into current  
git log --oneline → Concise history  
git reset --hard <commit> → Reset to commit (dangerous)  
git revert <commit> → Undo safely

# First-time identity (run once after install)

git config --global user.name "Your Name"

git config --global user.email "you@example.com"

(Optional helpful note to add once, near the first git push -u origin main: “Use -u only on the first push; later just run git push.”)

### 14. References

CS50 Git & GitHub: <https://cs50.harvard.edu/web/notes/1/>

W3Schools Git Tutorial: <https://www.w3schools.com/git/>

Official Git Docs: <https://git-scm.com/doc>