CS 450: Setup Instructions

DR. MICHAEL J. REALE

SPRING 2024

Overview (Part 1)

We will perform the following

- Install C++ compilers
- Install CMake
- Install and set up Visual Code
- Install the Vulkan SDK (which includes GLM)
- Install GLFW
- Install GLEW (needed for OpenGL)
- Install Assimp
- Install stb_image and stb_image_write

Separate instructions per OS follow for these steps.

Overview (Part 2)

We will also go over how to:

- Install and setup Git
- Create and clone your remote class project
- Commit your work and push to remote repo
- Work with branches
- Pull changes from original repo
- Work on and submit assignments

Windows Setup

Installing C++ Compilers

Download and install the "Build Tools for Visual Studio 2022": https://aka.ms/vs/17/release/vs_BuildTools.exe

- Under workloads, select "Desktop development with C++"

Installing CMake

Download the latest "Windows x64 Installer" for CMake: https://cmake.org/download/

Run the installer

- Make sure to "Add CMake to the system PATH for all users"

Installing and Setting Up Visual Code

Download and install Visual Code: https://code.visualstudio.com/

Open Visual Code and install the following extensions:

- "C/C++ Extension Pack" - Microsoft
- "GLSL Lint" – DanielToplak
- "Git Graph" – mhutchie

Installing Vulkan

You may have to uninstall any previous versions of Vulkan first!

Go to the LunarG website: https://vulkan.lunarg.com/

Download the developer tools for your OS → "Latest SDK"

Run installer

- Under "Select Components", make sure to check "GLM headers"

Check install location
(Windows should default to C:\VulkanSDK\<version number>)

- Run **Bin/vkcube.exe** to make sure drivers are up to date
- Make sure **Bin/glslangValidator.exe** and **Bin/glslc.exe** are present

Installing GLFW (Part 1)

While binaries do exist for GLFW, the more robust and flexible approach is to compile and install from source; download the **source package** here: https://www.glfw.org/download.html

Unzip the source code

Close ALL instances of Visual Code

Open Visual Code **as an administrator**

In Visual Code, open the unzipped folder

Select build kit

- View → Command Palette → "CMake: Select a Kit"
- Select "Visual Studio Build Tools 2022 Release - amd64"
  - If not found, try "CMake: Scan for Kits" first

Configure project

- View → Command Palettte → "CMake: Configure"

Installing GLFW (Part 2)

Change some of the configuration settings

- Under the Explorer view, open the file build/CMake**Cache**.txt
- Add the following anywhere in the file: **CMAKE_DEBUG_POSTFIX:STRING=_d**
- Find CMAKE_INSTALL_PREFIX and change the "GLFW" part to "glfw3"
- Save the file
- Configure again

Select "install" as the build target

- View → Command Palette → "CMake: Set Build Target"
- Select "install"

Installing GLFW (Part 3)

Build and install debug libraries

- View → Command Palette → "CMake: Select Variant"
- Select "Debug"
- View → Command Palette → "CMake: Build"

Build and install release libraries

- View → Command Palette → "CMake: Select Variant"
- Select "Release"
- View → Command Palette → "CMake: Build"

Installing GLEW (Part 1)

Similar to GLFW, we will compile and install GLEW from source; download **glew-2.2.0.zip SPECIFICALLY** from here: https://github.com/nigels-com/glew/releases

Unzip the source code

Close ALL instances of Visual Code

Open Visual Code **as an administrator**

In Visual Code, open the following folder INSIDE of the unzipped folder: **build/cmake**

Select build kit

- View → Command Palette → "CMake: Select a Kit"
- Select "Visual Studio Build Tools 2022 Release - amd64"
  - If not found, try "CMake: Scan for Kits" first

Configure project

- View → Command Palettte → "CMake: Configure"

Installing GLEW (Part 2)

Change some of the configuration settings

- Under the Explorer view, open the file build/CMake**Cache**.txt
- Add the following anywhere in the file: **CMAKE_DEBUG_POSTFIX:STRING=_d**
- Save the file
- Configure again

Select "install" as the build target

- View → Command Palette → "CMake: Set Build Target"
- Select "install"

Installing GLEW (Part 3)

Build and install debug libraries

- View → Command Palette → "CMake: Select Variant"
- Select "Debug"
- View → Command Palette → "CMake: Build"

Build and install release libraries

- View → Command Palette → "CMake: Select Variant"
- Select "Release"
- View → Command Palette → "CMake: Build"

Installing Assimp (Part 1)

Assimp must be compiled from source and installed; download the **Source code (zip)** for the latest release: https://github.com/assimp/assimp/releases

Unzip the source code

Close ALL instances of Visual Code

Open Visual Code **as an administrator**

In Visual Code, open the unzipped folder

Select build kit

- View → Command Palette → "CMake: Select a Kit"
- Select "Visual Studio Build Tools 2022 Release - amd64"
  - If not found, try "CMake: Scan for Kits" first

Configure project

- View → Command Palettte → "CMake: Configure"

Installing Assimp (Part 2)

Change some of the configuration settings

- Under the Explorer view, open the file build/CMake**Cache**.txt
- Set "ASSIMP_INSTALL_PDB" to OFF
- Set "BUILD_SHARED_LIBS" to OFF
- Save the file
- Configure again

Select "install" as the build target

- View → Command Palette → "CMake: Set Build Target"
- Select "install"

Installing Assimp (Part 3)

Build and install debug libraries

- View → Command Palette → "CMake: Select Variant"
- Select "Debug"
- View → Command Palette → "CMake: Build"

Build and install release libraries

- View → Command Palette → "CMake: Select Variant"
- Select "Release"
- View → Command Palette → "CMake: Build"

Installing stb Headers

The project I will provide already has the stb headers included.

However, if they are missing:

- Download stb_image.h: https://raw.githubusercontent.com/nothings/stb/master/stb_image.h
  - Right-click in browser, "Save As", and save as a header file (.h) into your **src/include** folder in your project
- Download stb_image_write.h: https://raw.githubusercontent.com/nothings/stb/master/stb_image_write.h
  - Right-click in browser, "Save As", and save as a header file (.h) into your **src/include** folder in your project

Mac Setup

Installing C++ Compilers

Mac should already have the Clang g++ compilers

Installing CMake

Download the latest "macOS 10.13 or later" .dmg for CMake: https://cmake.org/download/

Install Cmake and run it

Select "Tools" → "How to Install for Command Line Use"

Copy the second option into a terminal:

- sudo "/Applications/CMake.app/Contents/bin/cmake-gui" --install

Installing and Setting Up Visual Code

Follow the instructions outlined here under "Installation": https://code.visualstudio.com/docs/setup/mac

Open Visual Code and install the following extensions:

- "C/C++ Extension Pack" - Microsoft
- "GLSL Lint" – DanielToplak
- "Git Graph" – mhutchie

Installing Vulkan

Go to the LunarG website: https://vulkan.lunarg.com/

Download the developer tools for your OS → "Latest SDK"

Run installer

- Under "Select Components", make sure to check "GLM headers"

Check install location (Mac should default to /Users/<username>/VulkanSDK/<version number>)

- Run **Applications/vkcube** to make sure drivers are up to date
- Make sure **macOS/bin/glslangValidator** and **macOS/bin/glslc** are present

In the install location, run: sudo ./install_vulkan.py

Installing Homebrew and Other Packages

Install Homebrew (package manager for macOS):

- https://brew.sh/

Open a terminal and run the following:

- brew install glfw
- brew install glew
- brew install assimp
- brew install glm

Installing stb Headers

The project I will provide already has the stb headers included.

However, if they are missing:

- Download stb_image.h: https://raw.githubusercontent.com/nothings/stb/master/stb_image.h
  - Right-click in browser, "Save As", and save as a header file (.h) into your **src/include** folder in your project
- Download stb_image_write.h: https://raw.githubusercontent.com/nothings/stb/master/stb_image_write.h
  - Right-click in browser, "Save As", and save as a header file (.h) into your **src/include** folder in your project

Install Git

Introduction

Revision Control Systems

- Track revisions/versions of files
    - VERY frequently used to track the current status of code
- Often:
    - Identify WHO made changes to the files
    - Allow creation, deletion, and merging of “branches”
- Also called “Version Control Systems” (or VCS)

Common RCS:

- Git
- SVN
- Mercurial

Downloading and Installing Git

Download and install Git on your machine: https://git-scm.com/

- *Windows/Mac:* download and run the installer
- *Linux:* sudo apt-get install git

You should be able to open a terminal/cmd and type: git --version

Adding Name And Email

Windows: Open Git Bash

Linux/Mac: Open a terminal

Enter the following command, replacing "Your Name" with your actual name:

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

Enter the following command, using your email address:

```
git config --global user.email "youremail@yourdomain.com"
```

Making Your Class Project

Repository

Repository

- Contains files that you wish to keep track of
- May use folder hierarchy
- Can have **local** and **remote** repositories
  - **Local** = on your local machine
  - **Remote** = in the cloud, like GitHub or Bitbucket

I have created a GitHub repository for this class: https://github.com/PrimarchOfTheSpaceWolves/CS_450_2024_Spring

Creating Your Own Repository

Go to https://github.com/ and create an account

- Please use your SUNY Poly email address
- Remember that this will be part of your portfolio for potential jobs!

Create a new **PRIVATE** repository with the name

CS_450_<SITNET ID>

- E.g., CS_450_realemj

At the bottom, select "Import code"

Copy in my repository's URL:

https://github.com/PrimarchOfTheSpaceWolves/CS_450_2024_Spring

Add Professor as Admin

On your GitHub repository, go to Settings → Collaborators

Add me to your collaborator list:

- realemj@sunypoly.edu

Again, your repository should be

PRIVATE

Clone Your Repository

To create a copy on your local machine, you will need to clone the repository

Open Visual Studio Code (you should NOT have to run as administrator)

View → Command Palette → "Git: Clone"

- Paste in the address to YOUR remote repository
- Select a folder where you want your project folder to reside
  - Example: if you select “C:/Code”, then the project will be “C:/Code/CS_450_<SITNET>”
- Open the project

Remotes

A "remote" in repository terms refer to a place where code is stored (usually in the cloud, like on GitHub)

You will have two remotes when we are done:

- origin
    - Your repository on GitHub
    - Can be pushed to and fetched/pulled from
- upstream
    - The original repository you copied from (mine)
    - Can only be fetched/pulled from

Adding the Original Repository as a Remote

I will be adding in-class exercises AND any provided code for assignments to the original repository

To add that original repository as a remote:

- Using the Command Palette (View → Command Palette), run "Git: Add Remote"
- Enter URL for my repository: https://github.com/PrimarchOfTheSpaceWolves/CS_450_2024_Spring
- Enter the name "upstream"

When changes are made on the original repo:

- "Git: Fetch From All Remotes"
- "Git: Merge Branch" (more on this later)

One Project to Rule Them All

Once set up, your project will serve as the one project you need for:

- Assignments
- In-class Exercises

You should NOT need to make multiple projects!

Project Hierarchy

.vscode/

- **launch.json** – settings for debug runs
  - **args**: allows you to specific command-line arguments; **you will need this later to specify which 3D model to load!**

build/ - Contains currently compiled code and intermediate CMake configuration

- **If configuration/compiling becomes an issue, you can safely delete this folder and reopen the project.**

sampleModels/ - Contains sample 3D models and textures

src/

- **app/ -** Contains any main programs you will write
- **include/ -** Contains any header files you will write
- **lib/ -** Contains any shared source files you will write

shaders/ - Contains vertex and fragment shader files

CMakeLists.txt – defines how project is configured/built

Configuring, Building, and Running

Configure all projects:

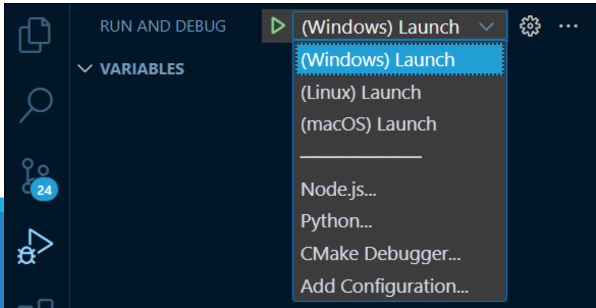
- View → Command Palette → "CMake: Configure"
- You will need to do this whenever:
  - CMakeLists.txt is changed
  - CMakeCache.txt is changed
  - You add a new header/source file
- You do NOT need to reconfigure if you are just modifying code.

Build desired target

- Each executable is a separate target
- View → Command Palette → "CMake: Set Build Target"
  - Should be able to choose "ALL_BUILD" for everyone
- View → Command Palette → "CMake: Build"

Run target (with command-line arguments from launch.json)

- View → Command Palette → "CMake: Set Debug Target"
- Go to the "Run and Debug" view on the side panel
- Select appropriate option for OS



Cleaning Project

To start over with CMake:

- View → Command Palette → "CMake: Clean"
- OR
- Just delete the whole build folder

Verifying Project

Make sure ALL of the following targets build and run:

- HelloWorld
  - Basic program to demonstrate command line arguments
- VerifyAssimp
  - Makes sure Assimp can load things properly
- VerifyVulkan
  - Makes sure Vulkan is working properly
- BasicOpenGL
  - Starting framework for future projects and exercises
  - WARNING: For macOS:
    - Change BasicOpenGL.cpp to use 4.1 instead of 4.3
    - Change Basic.vs and Basic.fs to use 410 instead of 430
    - (Search for "macOS" in those files)

Commit Your Work and Push to Remote Repo

Staging and Committing

In a repository, file changes are tracked

However, to save those changes, you need to:

- Stage the files
- Commit those files with a message

To save them on the remote repository, you need to:

- Push your commits

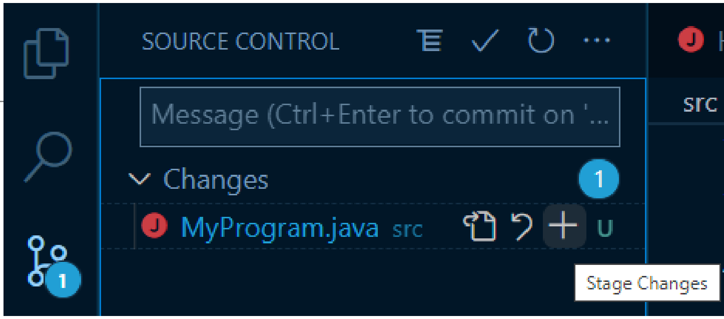
Staging and Committing in Visual Studio

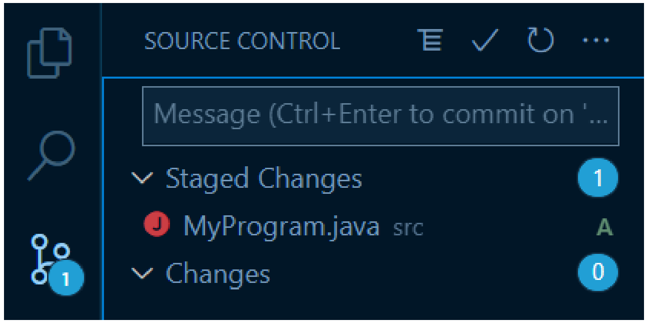
Let’s say you create a new file called “MyProgram.java”

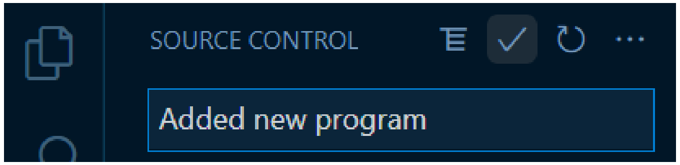
- Visual Code should fill in a default class

To stage file(s) and commit:

- Click the “Source Control” button on the left side
- Hit the plus sign next to all files you want to stage for this commit
- Add an appropriate message
- Click the checkmark to commit your changes







Added new program

Ignoring Files Forever

If you want to always ignore certain files (i.e., never included in git status), you can add them to a file called **.gitignore**

- One file per line
- Can use star wildcard: *.txt

Pushing Changes to Your Remote Repository

Commit any changes you want to keep from your work

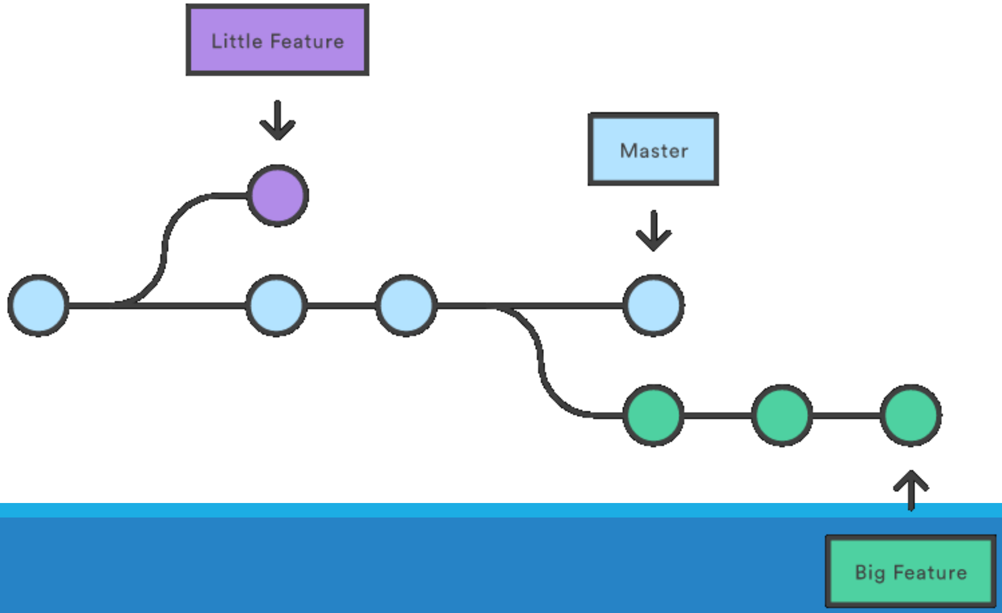
Using the Command Palette (View → Command Palette), run “Git: Push”

Introduction to Branches

Branches

Branch = separate path of development

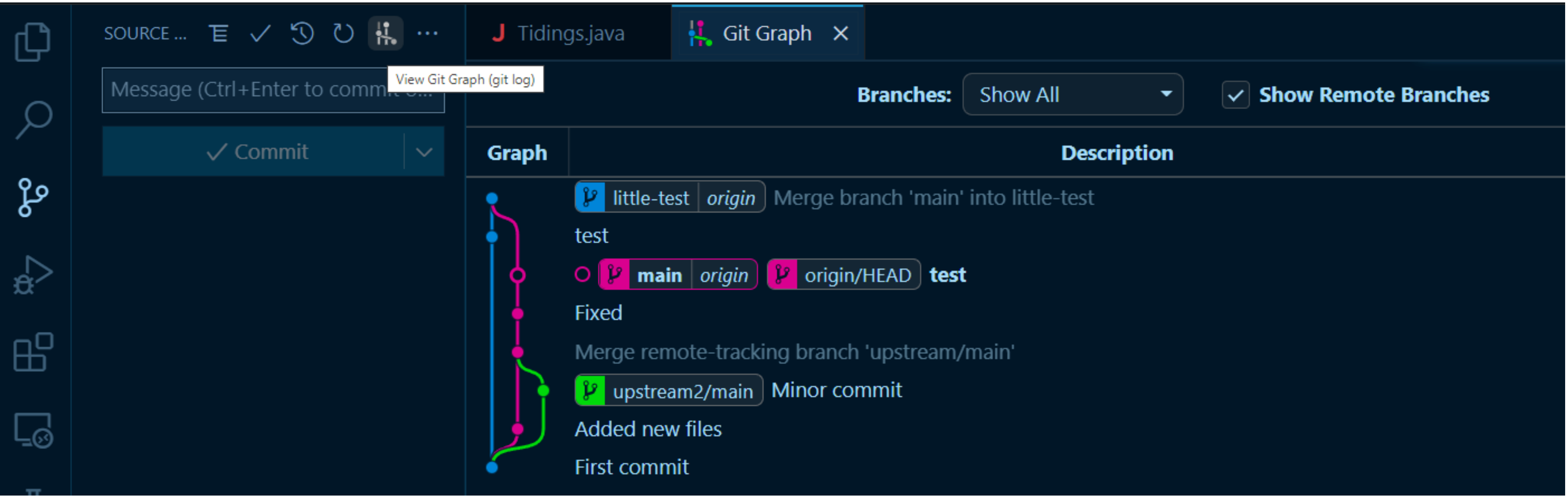
- Basically your own copy of files and commits that goes off on its own tangent
- Allows you to experiment / work on specific problem without messing with main branch



Viewing Branches

To see a nice graphical view of the branches:

- Click the "Source Control" button on the left side
- Click the "View Git Graph" button



Branches: Show All

☑ Show Remote Branches

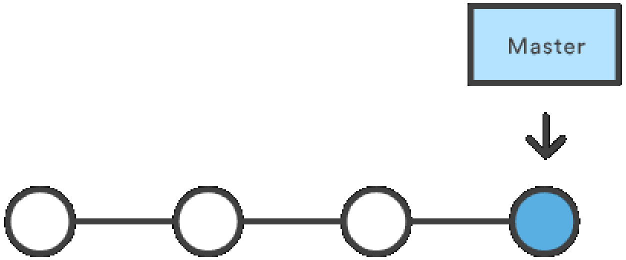
Graph	Description
	little-test origin Merge branch 'main' into little-test
	test
	main origin origin/HEAD **test**
	Fixed
	Merge remote-tracking branch 'upstream/main'
	upstream2/main Minor commit
	Added new files
	First commit

Git branches

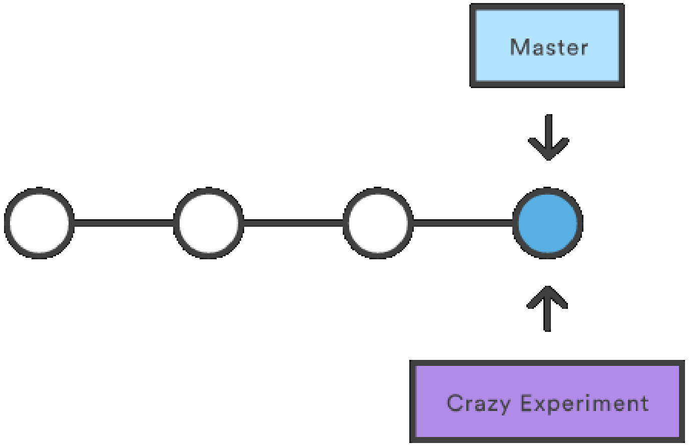
Branches in git = pointers to a commit

Example:

- Start with this:



- Create new branch: *git branch crazy-experiment*



Navigating Branches

To go to a different branch, we need to **checkout** a branch

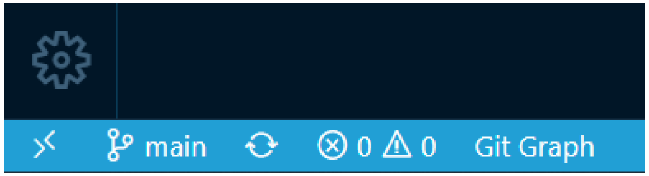
- WARNING: Sometimes creating a branch does NOT checkout the branch!
- **Make sure all changes are committed BEFORE switching branches!!!**

To create a branch off of current branch (and checkout)

- "Git: Create Branch"
- Enter name of new branch
- Check lower-left of window to see what branch you are on

To switch to a different branch:

- "Git: Checkout To"
- Choose branch to switch to



Merging

How do we add the changes we’ve made in our branch back into another branch (like the main branch)?

"Git: Merge Branch"

Select branch to merge FROM

- NOTE: Changes FROM this branch will be incorporated into CURRENT branch

Conflict Resolution

If there’s a conflict:

- Merge command will stop right before committing and tell you that conflicts are there
- Modify the files and then commit

Pulling Changes from Original Repo

In-Class Exercises and Test Programs

While on the branch you want to use:

- "Git: Fetch From All Remotes"
- "Git: Merge Branch" → upstream/main

Working on and Submitting Assignments

Working on Code and Submitting Assignments

Pull from upstream remote (original repository) and merge

Create and checkout a NEW branch

- In general, name of branch should reflect feature you are working on
- For assignments, the branch should be named **assignXX**, where XX is the zero-padded number of the assignment (01, 02, etc.)

Publish the new branch

Make all necessary code modifications and commits for assignment

- If additional files are requested (e.g., screenshots), make sure those are committed as well
- Push up to repository as necessary to save your work there

Merge your branch with YOUR main branch

Push up to YOUR repository

- **You MUST merge your final version into main AND push up to the remote repository BEFORE the deadline for each assignment!!!**