

Lab 4: Version Control

This lab will introduce version control with a tool called git.

Setting Up Git

There are two main steps in today's lab. First, you will just need to setup git and GitHub. Then, you'll practice using git.

Initializing VSCode for Git (all team members)

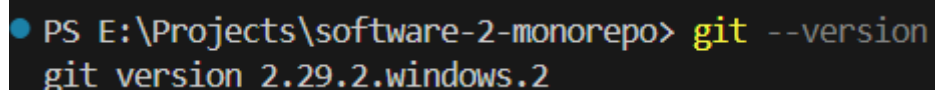
To ensure VSCode is setup for git, there are a couple things you need to do first, such as downloading and installing git and signing into GitHub. If any of the directions are unclear, feel free to follow [VSCode's tutorial directly](#).

Downloading and Installing Git

The very first thing everyone on the team should do is [download and install git](#). **Note:** It is possible that you already have this installed. To confirm, open a terminal (you can use the terminal in VSCode). Then, type the following:

```
git --version
```

If you see something like the following, you already have git.

A terminal window with a dark background. The prompt is 'PS E:\Projects\software-2-monorepo>'. The command 'git --version' has been entered and executed, resulting in the output 'git version 2.29.2.windows.2'.

```
PS E:\Projects\software-2-monorepo> git --version
git version 2.29.2.windows.2
```

If you see anything else, you do not have git. For Windows users, it should be straightforward to install git using the link above. If you're on macOS, the solution seems to be to install Xcode's command line tools, which ship with git. You can do that as follows:

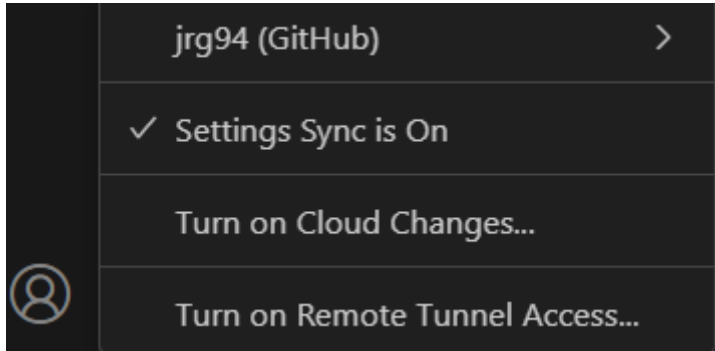
```
xcode-select --install
```

If that doesn't work, you can try to install homebrew, which relies on the command line tools from above. To do that, try downloading the `.pkg` from [Homebrew's latest GitHub release](#). Then, you should be able to run the following, as described on the [Download for macOS page](#):

```
brew install git
```

Signing into GitHub

Next, sign into GitHub through VSCode. To do this, click the profile icon in the bottom left of the sidebar. Then, click `Backup and Sync Settings`. From there, you should be able to sign into GitHub directly. If you're already done this, you will see `username (GitHub)` when you click on the profile icon. If you've done everything correctly, you should see something like the following:



Creating a Git Repo (only one member per team)

To share your projects and keep track of the history of the changes to the projects, you will need to set up a git repository. This is a simple task that only one team member must handle.

The repository for your team will reside on GitHub, an industry standard tool for hosting open-source software. Typically, there are two ways to approach this: one repository per project or one repository for all projects. In this course, we will be using the latter approach (i.e., the monorepo approach).

Creating a Repository

To create a repository, there are a variety of approaches, but one is by far the easiest.

1. Within VSCode, there is a source control icon that looks like a pair of branches. You can click it or press `CTRL + SHIFT + G` (or the Mac equivalent).
2. If done correctly, you will be greeted with two buttons: `Initialize Repository` and `Publish to GitHub`. Go ahead and click the second option: `Publish to GitHub`.
3. Next, VSCode will ask you if you want to publish a private repo or a public repo. For academic integrity purposes, **you MUST select private**. If you make the wrong choice, it can be changed on GitHub.

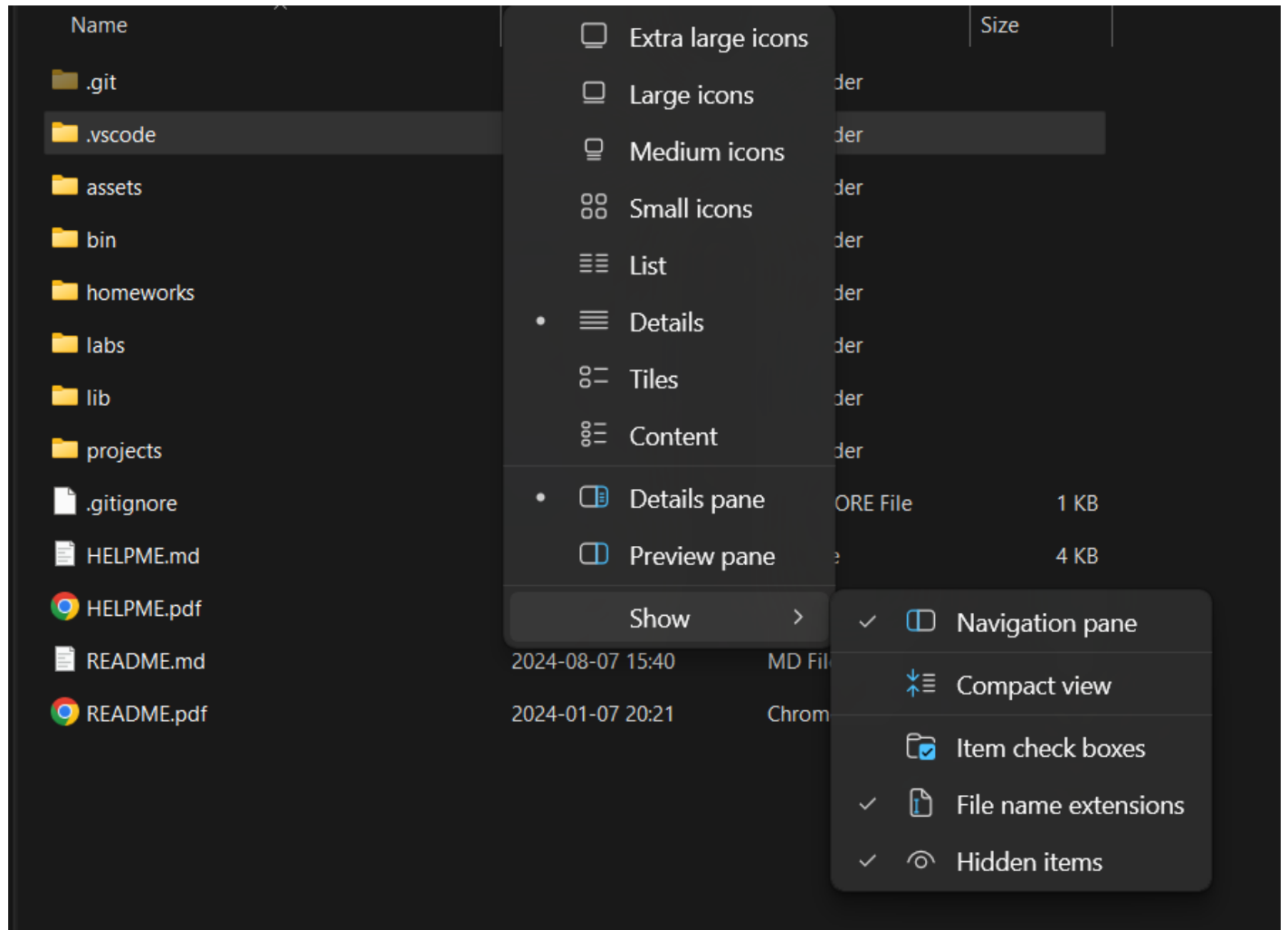
If all goes well, you will have a private repository on GitHub of your monorepo. Feel free to skip ahead to the [Sharing Your Repo](#) step.

When Creating a Repository Goes Wrong

If all does not go well, you may find that a repo is created (which you should see on the GitHub website itself), but there is no code. There are a few ways to fix this.

One option is to start over and try the steps above again. The way I would do this is by first deleting the repo on GitHub using the repo Settings tab. Then, I would recommend deleting the `.git` folder at the root of your repo. The problem is that the `.git` folder is often considered hidden by your operating system. Luckily, you can show hidden files fairly easily on both Windows and macOS.

On Windows, open the monorepo folder using Windows explorer, select `View`, select `Show`, and then select `Hidden items`. This should show the `.git` folder like in the following screenshot.



From there, you can just delete the `.git` folder. You may need to refresh the source control screen on VSCode for this to work.

On macOS, you can open the monorepo folder using Finder and enter the following key combination: `CTRL` + `SHIFT` + `.`. This will show hidden files. From there, you can also delete the `.git` folder. Alternatively, a slightly faster solution is to run the following command from the root of the monorepo using the terminal in VSCode:

```
rm -rf .git
```

If you don't want to start over or starting over does not work, we'll have to commit and push all the files manually. Consider following [the bonus guide](#) down below. The short answer is to try the following commands:

```
git commit -m "initial commit"
git push
```

If either of these commands fail, the errors should tell you what to do. For example, it's common for your git config to be missing your name and email. To add your name and email to git, you can run the following commands:

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

It's also common for the upstream branch to fail to be set. In that case, you will have to connect your local repo to your remote repo manually as follows:

```
git remote add origin REMOTE-URL
```

In this example, the remote URL is the URL you see on your repo on GitHub. It's typically in the form:

`https://github.com/username/repo-name.git`. At this point, you should be able to push using `git push`, and if this fails, git should give you the correct command to run.

Sharing Your Repo (same team member as last step)

Because your repo is private, you will need to add your teammate as a collaborator. To do that, you will need to open your repo on GitHub. Fortunately, you should have been prompted to do so after the last step by VSCode. If not, you can go to GitHub to find your repo there. VSCode may also have a "view on GitHub" feature somewhere, but I wasn't able to find it. One of the alternative ways to managing your repos, GitHub Desktop, has this feature built-in.

At any rate, once in your repo on GitHub, click on the `Settings` tab. In the sidebar, you will see the `Collaborators` tab. From there, you can click the `Add people` button and enter your partners username or email address. If you're having trouble finding this page, the URL path is as follows: `https://github.com/username/repo-name/settings/access`.

Cloning a Repo (other team member)

Assuming the first teammate set everything up correctly, the next step is to clone the repo to your computer. Again, there are a lot of ways to do this, but to keep things as easy as possible, we'll make use of VSCode.

1. Press `CTRL` + `SHIFT` + `P` (or the Mac equivalent) to open the command palette.

2. Next, type `git clone` and select the "Git: Clone" option.
3. It should give you the option to clone a project from GitHub. Click that option.
4. Next, a list of repos will be presented to you. You can scroll through this list or just search for `software-2-monorepo`. Selecting a repo will prompt you with a location on your system for cloning.
5. VSCode should then prompt you to open the newly cloned repo. Click `Open`.

At this point, you will both have the repo open in VSCode. For any teammates performing this step, you will now have two versions of the software-2-monorepo on your system. If you'd like to only maintain one, I recommend copying over any files you'd like to keep (e.g., the first few homeworks) and deleting the old one. In the future, I may streamline this process.

Working Together

Now, the hard part: learning to use git. To do that, we'll be using the `Queue3.java` file in the `src` folder of the 04-version-control lab. For the purposes of the lab, we will be working directly off of `main` (note: don't worry if this means nothing to you at the moment).

To start, pick a teammate to make some changes to the `Queue3.java` file. When the lab says swap, move to the next teammate.

Implement Enqueue (pick a teammate to do this)

1. Complete the body of `enqueue()` and save the file.
2. Open the source control panel (e.g., using `CTRL + SHIFT + G`).
3. Git has this concept of staging, which allows you to pick and choose which files you want to commit. You can manually select the files in VSCode using the `+` icon next to each file. Alternatively, you can skip to step 4.
4. Write the following commit message in the box: `implemented enqueue` and click commit. If you did not do step 3, you may be prompted to stage all changes. Click `Yes`.
5. When you commit files, they are only committed locally. To ensure the changes are pushed to GitHub, the `Commit` button should have transformed into a `Sync` button. Click it.

Implement Dequeue (pick another teammate to do this)

6. Complete the body of `dequeue()` and save the file.
7. Open the source control panel (e.g., using `CTRL + SHIFT + G`).
8. See step 3 above.
9. Write the following commit message in the box: `implemented dequeue` and click commit.
10. Again, this change is only committed locally. When you click sync, git is going to attempt to merge the two versions of the file. In this case, things should go smoothly.

Getting Latest Changes (go back to first teammate)

11. The previous teammate should be up to date. If you want to be up to date too, you should open the source control tab and select sync. If you don't have the option to sync, try refreshing.

Causing a Conflict (both teammates)

12. Now that you're both up to date, both of you should implement the `length()` method: one using the `length` method of entries and the other using a loop. We want two different solutions.
13. Both of you should commit your solution, and both of you should sync.
14. Depending on the order of syncs, one of you will be left with a nasty merge conflict. This occurs when two people edit the same line. Git doesn't know which line you want, so it flags it as a conflict. You must manually go through the code and fix the conflict. Luckily, VSCode highlights the conflicts directly in the file for you. You will be given four options for each line in conflict: accept current change, accept incoming change, accept both changes, or compare changes. It will not always be obvious which change is correct, so you will have to make an informed decision about what changes you want.

Note: lines of code not in conflict will be merged normally, even if you don't want them. For example, if the person who wrote the loop syncs last, their loop will likely stay intact. They will have to choose between the two different return statements. Of course, they should amend their solution to remove the loop.

Verifying the Code

At this point, you can run the test files like normal to ensure your implemented the kernel correctly. If everything works, you're done! In the future, I might include a way to run the tests automatically on GitHub as well. If that's something you're interested in, check out GitHub Actions.

Bonus: a Better Workflow

Unfortunately, one lab isn't enough time to teach you how git works properly. You will likely spend most of this semester only scratching the surface of the tool. That said, my general advice is to make use of branching for every single project. To do that, follow these steps:

1. Have one teammate create a branch off of main. This can be done by opening the command palette using the usual command (i.e., `CTRL` + `SHIFT` + `P`).
2. Search the command palette for `Git Branch` and select `Git Branch From` .
3. Select `main` and then type the name of your branch. For example, if you want to start project 2 with your teammate, name the branch `project-02` .
4. From the source control panel (i.e., `CTRL` + `SHIFT` + `G`), select `Publish Branch` . That way, your team can access it.
5. Now, treat this new branch just like you would `main` . Commit and sync to it until you're finished with the project.
6. When the project is complete, you will create a pull request to GitHub. Normally, this is very easy from GitHub Desktop. Instead, VSCode doesn't seem to have a feature for it (without installing another extension). As a result, head over to your repo page on GitHub. There should be a button at the top of the page asking if you'd like to make a pull request. You can click that or head to the pull requests tab and click `New pull request` . From there you can select a branch, in this case `project-02` .
7. Give your pull request a title like "Completed Project 2" and describe what you did.
8. Next, let your teammate know that the project is complete and give them a chance to review the code. Your teammate should be able to review the code and make comments. If they're satisfied, they can

approve of it.

9. Once approved, merge the pull request into main. My preference is to use the squash and merge feature, which makes the history a bit cleaner. However, any merge should be fine.

If you follow this process for every project, you'll be using git in a much more professional manner. Please consider this workflow even though we will never see you do it.

Bonus: Git Commandline

At some point, you will break git. As a result, it's a good idea to get familiar with the commandline. [Here's a guide from GitHub directly](#) on how to create and push a repo from the commandline. Think about it like looking under the hood of VSCode's implementation.

Here are a few additional commands I would memorize:

1. `git init -b main` : converts the current folder to a git repo with a branch names "main"
2. `git status` : gets the current status of the git repo; probably the most useful command
3. `git add .` : adds everything that is not ignored by the `.gitignore` to be committed
4. `git commit -m "your message"` : makes a local commit with a commit message
5. `git pull` : pulls the latest changes to the current branch
6. `git push` : pushes the latest changes to the current branch (always pull first)

There are probably hundreds of git commands, but these are enough to make you dangerous.