We have allowed you to comment and suggest on the Google Docs (ping us on the message board if you can't) if you would like to provide any feedback (anything small or big such as a typo, grammar mistake, suggested change, etc. is welcome).

CSE 332: Data Structures and Parallelism

Setting Up Your CSE 332 Environment

This document guides you through setting up IntelliJ for CSE 332 in various parts. If you run into any problems or questions when setting up or using any of these plug-ins, feel free to ask on the message board or during office hours!

Use the Document Outline feature on Google Docs to navigate around the document.

Part 1: Downloading Tools

The first thing you should do is download and install the tools needed to work locally.

1) Download **OpenJDK 11** from AdoptOpenJDK.
    - However, if you have an **Apple M1-based MacBook** (November 2020 and later), download OpenJDK 11 from Azul Systems (direct link). If you do not install this instead, IntelliJ will try to use an Intel x86-64 based JDK running in compatibility mode and will perform significantly slower.
2) Download Git, our version control software.
    - In macOS and Linux, this might already be installed. You can type `git` in a terminal to check.
3) Download IntelliJ IDEA, our IDE.
    - If you have an **Apple M1-based MacBook**, take care to choose the ".dmg (Apple Silicon)" download. It has significant performance implications.
    - As a student, you can get a free education license and can access either the Community or Ultimate Edition of IntelliJ IDEA. If you don't want to register for a JetBrains account, the free Community Edition is totally sufficient for this course.
    - Even if you already have IntelliJ installed, we encourage you to update to the latest version.

Part 2: Setting Up GitLab

We will be using CSE GitLab to submit homeworks and give feedback. GitLab is a web-based git hosting service that is similar to GitHub but hosted locally by CSE.

You will need to learn basic git to be able to work on and submit your homework, but it's okay if you don't know what it is or if you've never worked with it before! There's a brief introduction to Git at the end of this document.

These steps are error-prone and hard to get right the first time. Please ask for help if you get stuck!

Locating your terminal

For the rest of the document, when we mention "open a terminal", we mean:

- Windows: Open **Git Bash** after you install Git. Please note that this is not the WSL (Windows Subsystem for Linux) terminal.
- macOS: From your Finder, go to Applications -- Utilities, and open **Terminal**.
- Linux: Use any terminal emulator of your preference. On CSE computers, hitting the Windows key and typing "Terminal" into search will work.

Generating an SSH key

Before you can clone from GitLab, you will need to create an *SSH key*. SSH is a protocol supported by git to access GitLab securely and without password authentication.

You might be able to skip this entire section if you have already done this for another course. To check if you already have a key generated:

1. Open a terminal and run the following command:

```
ls ~/.ssh
```

2. If you see a file called `id_ed25519.pub` or `id_rsa.pub`, that is your public SSH key. You can now proceed to the next section. Otherwise, continue on.

To generate a new SSH key:

1. Open your terminal and run the following command:

```
ssh-keygen -t ed25519
```

2. Next, you will be ation.
3. Once the path is decided, you will be prompted to input a password to secure your new SSH key pair. It's not required. If you don't want to enter your password every time you use Git, you can skip creating it by pressing **Enter** twice.

Adding SSH to your GitLab account

If you are using the CSE GitLab for the first time, you will need to submit your SSH key. To do this, follow these steps:

1. Open a terminal and execute (replace with `id_rsa.pub` if needed):

   ```
   cat ~/.ssh/id_ed25519.pub
   ```

   This will print out the public key that you just generated.
2. Visit GitLab's SSH key management page by going to https://gitlab.cs.washington.edu/-/profile/keys. Paste your key into the box, and the rest of the fields should be filled in automatically. Press **Add key**.
3. In the terminal, run `ssh -T git@gitlab.cs.washington.edu` to ensure that your key is correctly set up. You should receive a welcome message and should not be prompted for a password.

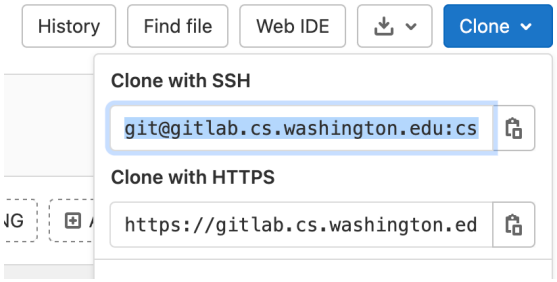
Creating The IntelliJ IDEA Project

Each project in the course will have its own project in IntelliJ. We will create the project in IntelliJ by cloning your assigned `git` repository from `gitlab` which contains the starter code (this part is irrelevant before being assigned a `git` repository). To do this, follow these steps:

1. If no project is currently opened, choose **Get from VCS** on the Welcome screen. Otherwise, from the main menu, choose **File | New | Project from Version Control**.
2. In the Clone Repository dialog, specify the URL of the remote repository you want to clone. It should look something like this (and should start with git@ instead of https):

   git@gitlab.cs.washington.edu:cse332-22wi-students/p1-pikachu.git

   You can find this URL by going to your repository's page on GitLab, and clicking the blue **Clone** button. Click the clipboard in the **Clone with SSH** section:



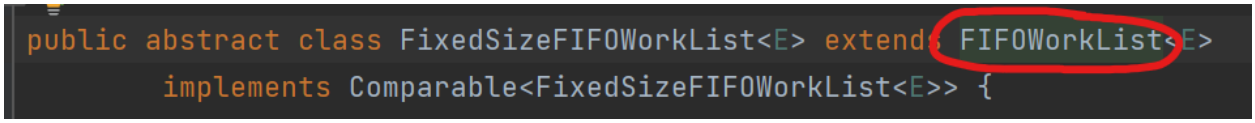
3. In the **Directory** field, specify the path where the folder for your local Git repository will be created into which the remote repository will be cloned.
    - Take care to not put your Git-managed project files in a folder synced using another tool, like OneDrive or Google Drive. They are known to cause corruption and other performance weirdness.
4. Click **Clone** and click **Yes** in the confirmation dialog.
    a. Popups may appear asking you to open the project after check-out. Answer in the affirmative (you might see either **Yes** or **Trust**).
    b. Another pop-up may appear notifying you that the SDK is not set-up or configured, click **Yes** and the SDK will be set up later.

Part 2.5: IntelliJ Tips

This is just a compilation of IntelliJ tips some students have found useful, sorted by arbitrary "usefulness":

- Probably one of the most useful feature for navigating through our codebase is ctrl (or command) clicking on symbols such as class names: https://www.jetbrains.com/help/rider/Navigation_and_Search__Go_to_Declaration.html#eb9663d

  Want to go to `FIFOWorkList` from `FixedSizeFIFOWorkList`?

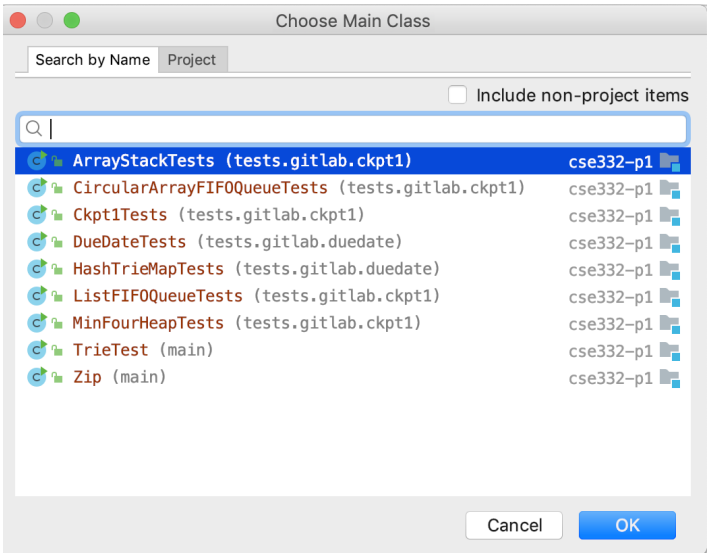
  Just ctrl (or command) + left click on `FIFOWorkList` at the top.
- Pressing Shift twice will open the "Search Everywhere" menu that will help you navigate large codebases (such as the projects) easily: https://www.jetbrains.com/help/idea/searching-everywhere.html

  Useful for finding that one file you just can't seem to ever find.
- You can refactor (fancy rename) elements of your code such as variables or method names to help with code organization rather than manually changing everything: https://www.jetbrains.com/help/idea/refactoring-source-code.html#refactoring_invoke
- You can search and replace text in the entire repository in IntelliJ: https://www.jetbrains.com/help/idea/finding-and-replacing-text-in-project.html
  This is useful for, say, finding an accidentally misplaced `System.out.println` in your project.

Part 3: Running Tests in IntelliJ

Here are a few ways to run your tests in IntelliJ.

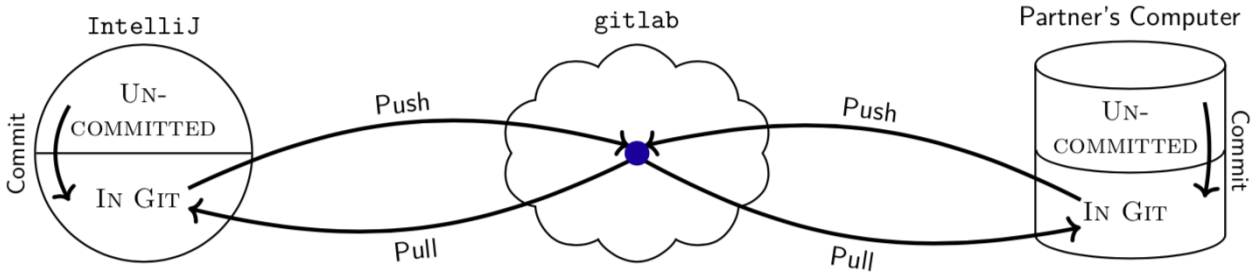
- Click the **green arrow** near the line numbers and main method
- Right click the Test File name and select "Run TestFile.main()"
- Click **Add Configuration** on the upper left side corner. In the "Run Configuration" Pop-up, click the **+** button and select **Application**.



Specify the main class as the test file you want to run. You can click ... on the right side to select from all the test file options.

Part 4: Using Git

`git` is the *version control system* (VCS) underlying Gitlab. Most real projects are kept in a VCS to avoid losing data, and for the ability to easily revert code back to older versions. Another major reason VCSs are important is that they allow you to effectively work together with other people. They allow you to combine (“merge”) several different versions of your codebase together.



As shown in the diagram, there are several major actions you can do with respect to your git repository:

- Commit: A “commit” is a set of changes that go together. By “committing” a file, you are asking `git` to “mark” that it has changed. `git` requires that you give a message summarizing the effect of your changes. An example commit message might be “Add error handling for the empty queue case in ListFIFOQueue”.
- Push: A “push” sends your commits to another version of the repository (in our case, this will almost always be `GitLab`). If you do not push your commits, nobody else can see them (including Gradescope)!
- Pull: A “pull” gets non-local commits and updates your version of the repository with them. If you and someone else both edited a file, `git` will ask you to explain how to merge the changes together (this process is called “resolving a merge conflict”).

Using Git in IntelliJ

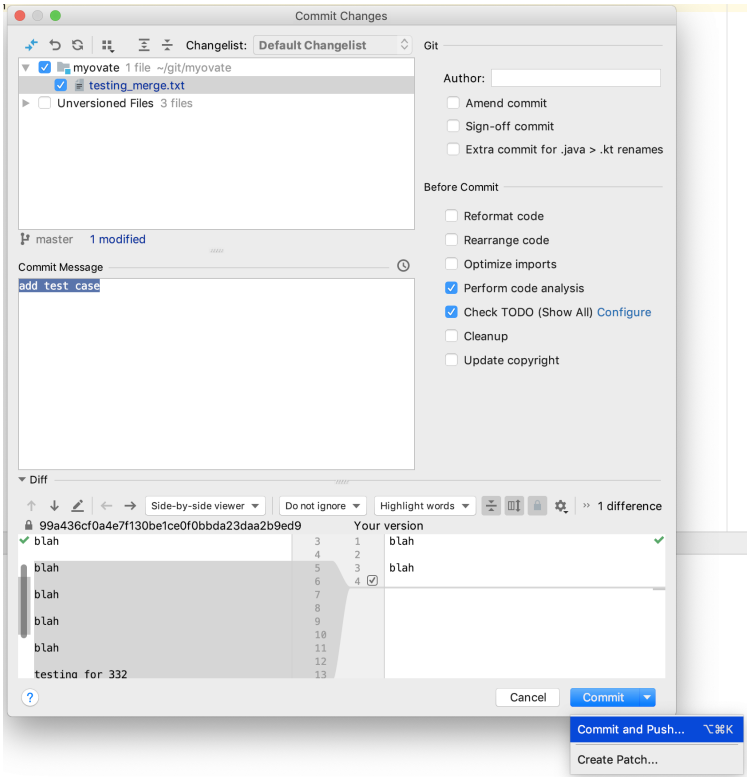
IntelliJ provides a GUI for all of the git operations. We now explain how to handle a git workflow in IntelliJ.

Committing and Pushing

After making changes to, adding, or removing files, you must commit and "push" your changes to git. This step will cause git to record your changes to the repository, so that

your changes are backed-up and available to other people working on the repository, or to you when working on a different computer system.

1. Click the green checkpoint in the upper right corner or in the main menu **VCS | Commit**

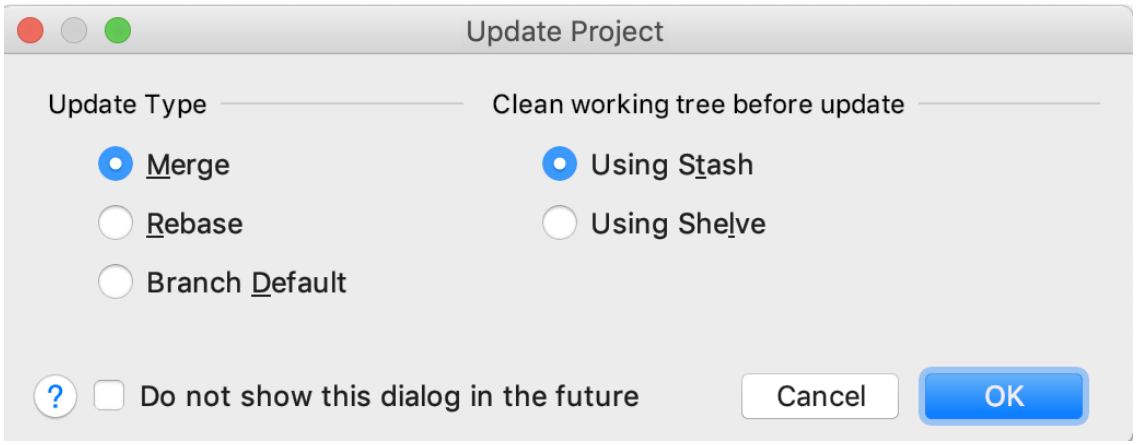


2. Select all the files to commit (all the files you changed should be selected by default) and add an appropriate commit message.
3. Press the **down** arrow next to the "Commit" button, and select **Commit and Push**
4. Note if you accidentally press the "Commit" button, push manually by going to **VCS | Git | Push**.

Pulling

There are two reasons to pull: (1) you want to get the changes your partner made, or (2) you want to push your changes, but they were rejected because of a conflict.

1. To pull in IntelliJ, from the main menu, choose **VCS | Update Project** or click on the **blue arrow** on the upper right side corner.



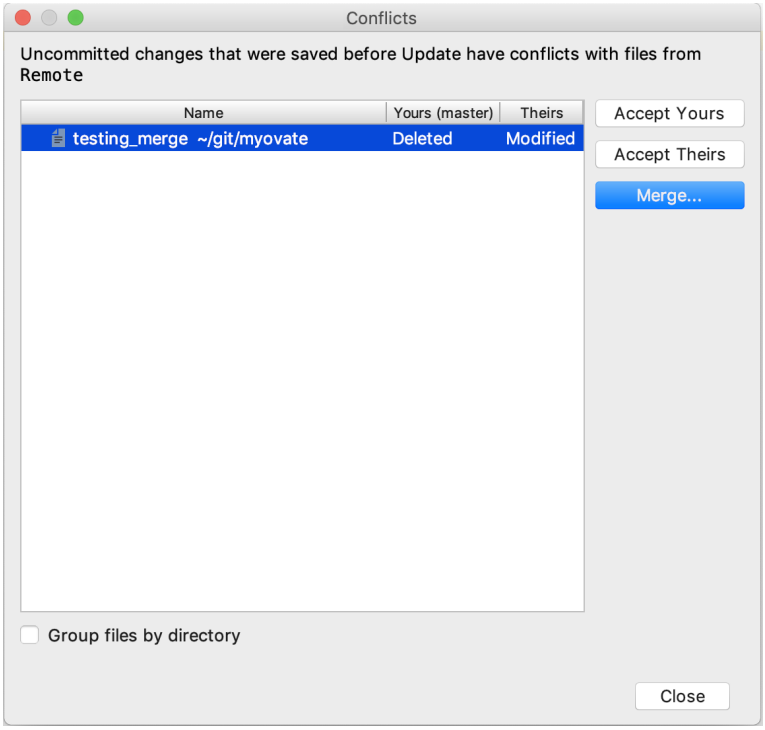
2. The following window should appear. The default selections should be **Merge** and **Using Stash**, if not, change ”Update Type” and ”Clean working tree before update” to them respectively.

The result will either indicate that you pulled cleanly or that there is a merge conflict. If you have a merge conflict, then a conflict pop-up will appear, which means you have uncommitted local changes which conflict with changes on GitLab. Refer to the next section, Merging a Conflict.

Merging a Conflict

If you have any items that have a conflict and you select the **Merge...** option, there are two ways to resolve the conflict: (1) Use the *merge tool* that appears (2) Exit out of the *merge tool* and resolve the conflicts manually

1. Using the merge tool



   a. If you want to use your version of the file, select **Accept Yours**.
   b. If you want to use your partner's version on the file, select **Accept Theirs**.
   c. If you want to merge your version and your partner's version of the file, select **Merge...**. The *merge tool* should appear. Refer to the IntelliJ Help to resolve the conflict using the *merge tool*.

2. Resolving the conflict manually
   When `git` detects a file conflict, it changes the file to include both versions of any conflicting potions in this format:

```
<<<<<<< filename
YOUR VERSION
=======
REPOSITORY'S VERSION
>>>>>>> 4e2b407... -- repository version's revision number
```

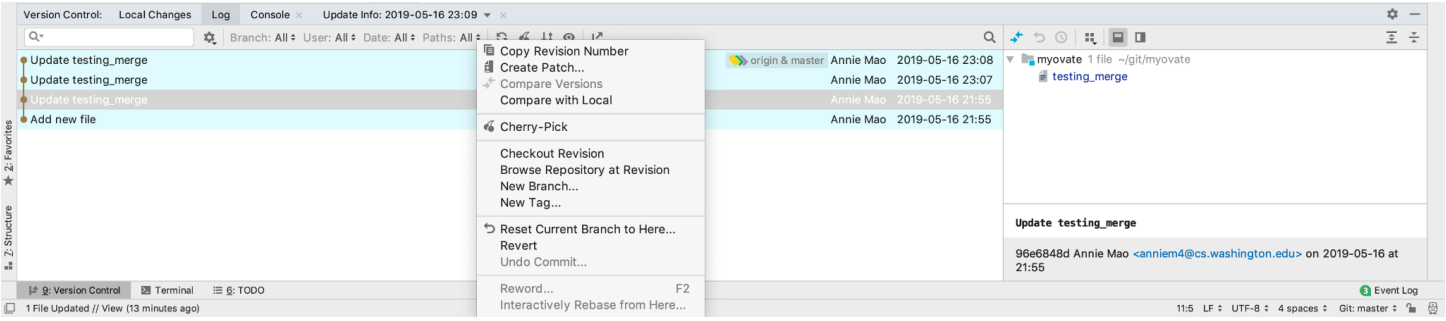
   Each conflicting file will be colored in Red. For each conflicting file, edit it to choose one of the versions (or to merge them by hand). Be sure to remove the

<<<<<<<, =======, and >>>>>>> lines. (Searching for <<< until you've resolved all the conflicts is generally a good idea.)

Don't forget to commit and push these changes as you normally would once you have made these edits, so that you can tell git that you have resolved the conflicts. Note that if there are still conflicting parts of the file (<<<<<<<, =======, and >>>>>>> lines still remaining), the commit will fail.

Revert a commit

1. Open the **Version Control** tool window at the bottom left corner
2. Select the **Log** tab



3. Locate the commit you want to revert, right click, and select **Revert**
4. Select the files you want to revert, and deselect the files you do not want to revert
5. Press the down arrow next to the "Commit" button, and select **Commit and Push**

Reset a Branch to a Specific Commit

1. Open the **Version Control** tool window at the bottom left corner
2. Select the **Log** tab
3. Locate the commit you want to revert, right click, and select **Reset Current Branch to Here**
4. In the Git Reset dialog that opens, select **Mixed** and the index to be updated and click **Reset**.

Submitting Your Final Version

In some courses, you are asked to "tag" your final commit. We will not ask you to do this. Instead, when you submit to Gradescope, it will automatically use your final commit.