# Working with Git and the JSL

Git is a version control system. The JSL is based on a Git repository. There are two major methods for getting the JSL source code from the repository.

1. Cloning the repository
   1. Cloning the repository occurs when you follow the instructions on the repository. This gives you a copy of the repository that is local on your computer that has all of the code necessary to set up a Gradle based project.
   2. It automatically makes a “connection” between the main repository and your local repository. You can use this connection to update from the main repository when there is an important change.
   3. If you were given special permissions, you could actually push your code back to the main repository. Don’t worry. You do not have those permissions.
2. Forking the repository
   1. A fork is a complete copy of the repository. It is almost exactly like a clone, except that it is independent of the main repository. There is not an underlying connection between the main repository and the fork, unless you take special action.
   2. You have total ownership of the fork. You can do whatever you want to the fork and it will not affect the original repository.

This document overviews some basic practices that would facilitate you keeping your version of the JSL up to date. You can do all of these practices using IntelliJ or the command line. However, I use the (free for non-commercial use) program called GitKraken to help me manage repository work. For more on working with Git, see the useful tutorials[[1]](#footnote-1) that you can readily find on-line.

# Coding Practices

In order to minimize conflicts with the main repository, I recommend the following practices. These practices are useful when cloning or forking, but more so for the case of cloning. Note that you do not really have to do practice 1 and 3, but practice 2 is critical to ensure that there will never by any conflicts between your code files and the files within the JSL. While git can be used to view and manage the change to files, I suggest these practices to simplify this entire exercise.

1. After downloading and setting up the JSL using the instructions provided on the repository, your first act should be to make a new branch. I recommend using your UA user id to provide a unique name for the branch. You will treat this branch as your “master” branch for all of your work. In this way, any changes that are made to the original master branch cannot affect your work unless you explicitly take action to incorporate those changes.
2. Create a package at the same level as the ex, jsl, and test packages within your dedicated branch. I recommend using your UA user id to provide a unique name for the package. Place all of your work within that package, including additional packages. Organize your sub-packages in a way that makes your coding work for you. This step additionally isolates your work from changes within the master branch, even when you explicitly take actions to utilize those changes. This prevents unnecessary management of file conflicts.
3. When you develop code based on steps 1 and 2, you can (and should) commit changes to your working branch. A git commit is like a “save point”. This will allow you to manage your code locally using the version control system. Feel free to branch off of your branch and operate accordingly. You can and should have many commits along your branch.

# Updating Your branch

Suppose an important change to the JSL master branch has occurred and you want to incorporate those changes into your working branch, what should you do? Assuming that you are practicing the previously mentioned practices, your work is very straight forward. Basically, you pull the updated master branch to your local repository and then merge the updated master branch into your working branch. You can do this work either from the command line or from a Git client such as IntelliJ or GitKraken. I will illustrate the commands through the command line.

1. Stage and commit any changes to your local branch. Make sure that your local branch is the active branch. Open up the terminal and make sure you are in the directory holding your repository. Within IntelliJ, this is the default behavior when opening a terminal from within IntelliJ. This step ensures that all your work is committed before proceeding.

git checkout branchName // switches to the branch and checks it out, branchName is the name of the branch.

git commit -a -m ‘the commit message’ // adds all changed files to the commit and commits them

1. Now switch to the master branch, pull changes from it, and commit those changes

git remote -v // this shows your remotes, you should see origin referencing the JSL repo

git checkout master // switches to the master branch

git fetch origin // origin is the name shown when you did remote -v

git commit -a -m ‘updated the master’ // adds all changed files to the commit and commits them to your local master

Now your master is synced with the remote master and contains any of the changes from the remote repository. You can now merge those changes into your working branch. To merge from one branch, you must check out the branch that you want to merge into and then merge from the other branch.

git checkout branchName // changes to your working branch

git merge master // merges from the master into your current branch

git commit -a -m ‘updated my branch from the master’

After doing all of this work you should re-build your project through IntelliJ to ensure that everything compiles appropriately.

## Updating Your Fork

If you forked the JSL, then your work is more complicated. You must first establish a remote to the original repository for your fork.

git remote -v // this shows your remotes

git remote add upstream [https://git.uark.edu/rossetti/JSL.git //](https://git.uark.edu/rossetti/JSL.git%20//) add a remote to the original JSL

git pull upstream master // pulls from the remote you just defined

git push origin master //pushes to your forked repo master, now your fork should be updated

Once your master is updated, you can then merge from your master into any working branches as need. Please see this reference for further information: <https://gist.github.com/Chaser324/ce0505fbed06b947d962>

1. <https://git-scm.com/book/en/v1/Getting-Started-Git-Basics> [↑](#footnote-ref-1)