**CSC 312 Collaboration with Git**

In CSC312, you and your teammate will be using below basic collaboration scenario without branches. (Although various branching strategies are useful to know, to keep this course focused on Algorithms topics itself, branching and making pull requests will be outside the scope of this class. But you are certainly free, not required, to explore them on your own and incorporate them in the projects.)

**[Scenario 1: Basic Sharing without Branches]**

We have **Alice** and **Bob** collaborating. Say **Alice** made a change to Hello.java in the project. **Alice** should commit and push the code to the team repository.

She does it by doing

$git add --all

$git commit -m “Alice committing v1”

$git push

Then **Bob** needs to get the latest code from the team repository.

He does it by doing

$git pull

**[Scenario 2: Merge Conflict]**

When two or more people work on the same files, conflicts are bound to occur. Version control will help notify us when there are conflicts. It will be up to the humans to sort out which changes to retain. (The best practice is avoiding merge conflict by always git pulling before you make any changes.)

Let’s say **Alice** modifies line 10 of Hello.java and pushes changes to GitHub:  
$ git commit -am "added a line in local copy and pushed to remote"  
$ git push origin master

Now, **Bob** modifies her local file WITHOUT first updating it (that is WITHOUT pulling the repo – BAD PRACTICE!!) from GitHub:  
Bob also modifies line 10 of Hello.java and commits the changes locally:  
$ git commit -am "added a line in local copy"

This will go through

But when **Bob** tries to push, by typing below,  
$ git push

Git will not allow this because there were changes to the same line in the two files:

To resolve the conflict, **Bob** needs to **pull** the changes from GitHub, **merge** them into your local copy, and then **push** it back to GitHub  
$ git pull origin master

Git tells us there is a conflict and tells you the file it’s in.  
When **Bob** looks at the file, Git has put some new info in our file:

<<<<<<<<<<<<< HEAD  
Alice typed something here  
========  
Bob typed something else here  
>>>>>>>>

**Bob** needs to manually make the changes which is the accepted version by you and your collaborator:

Then, to finish mergining, you need to **add**, **commit**, and **push** your changes back to GitHub:  
$ git add tenlines.txt

You can verify the status of your repo first, then commit and push:  
$ git status  
$ git commit -m "Merged changes from GitHub"  
$ git push origin master

Git keeps track that a conflict has been resolved and what was merged into what, so when **Alice** who made the first changes pulls from GitHub, she doesn’t have to fix things and merge again.

**Alice** should pull and and she will see new version of the file that **Bob** resolved:  
$ git pull