ENSF 607 – Software Design and Architecture I
Fall 2020

Introduction to version control systems – Working with Git

Due Date
Demo your Exercises at the start of your lab on Friday September 18th

The objectives of this exercise are:

- to get familiar with version control systems.
- to learn how to work with Git repository as a team.

This assignment **MUST** be done in groups of 3



Exercise - 1: First Git Project

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work. Listed below are the functions of a VCS:

- Allows developers to work simultaneously.
- Does not allow overwriting each other's changes.
- Maintains a history of every version.

Some of the well-known VCSs are Concurrent Versions System (CSV), Git, Apache Subversion (SVN), Team Foundation Server (TFS).

VCSs are of two types:

- Centralized version control system (CVCS)
    - Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. But the major drawback of CVCS is its single point of failure, i.e., failure of the central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all. In a worst case, if the disk of the central server gets corrupted and proper backup has not been taken, then you will lose the entire history of the project. Here, distributed version control system (DVCS) comes into picture.

- Distributed/Decentralized version control system (DVCS)
    - DVCS clients not only check out the latest snapshot of the directory but they also fully mirror the repository. If the server goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository. Git does not rely on the central server and that is why you can perform many operations when you are offline. You can commit changes, create branches, view logs, and perform other operations when you are offline. You require network connection only to publish your changes and take the latest changes.

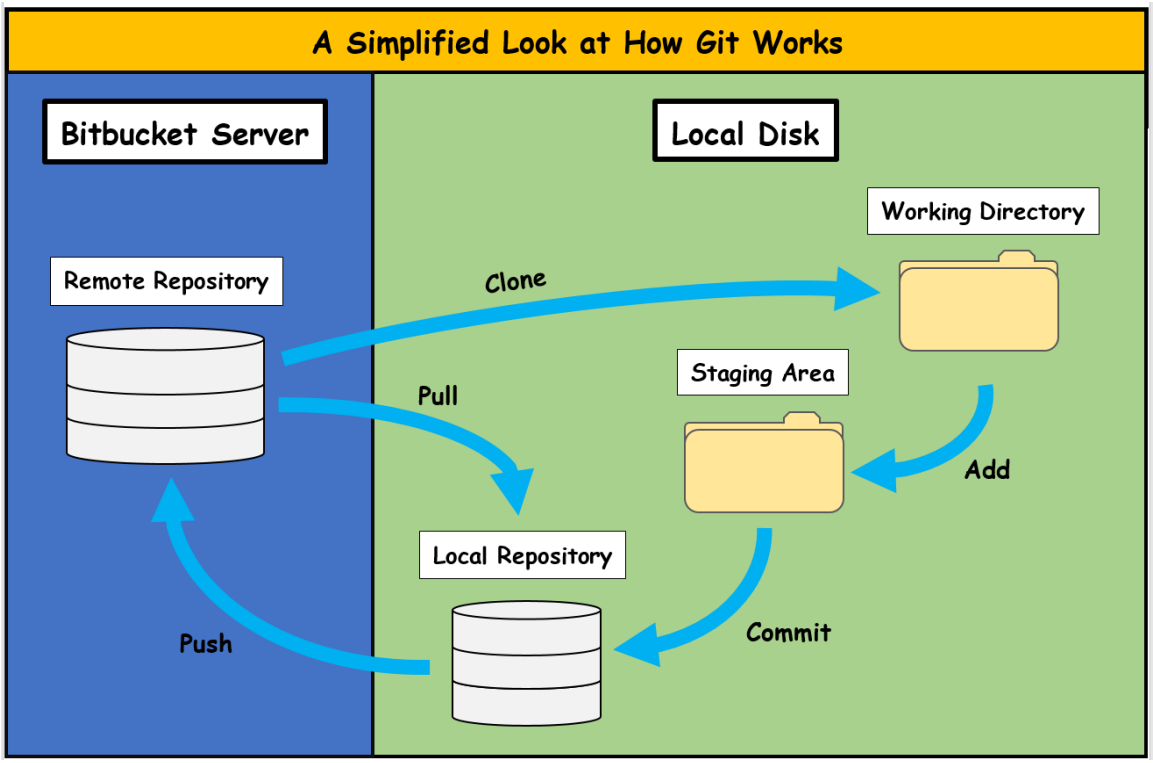
In this assignment, we will concentrate on Git version control system. There are many online text and video resources that you can use for setting up Git and learning its commands. You can use the tutorials in Atlassian website to learn about Git and working with Atlassian Bitbucket:

https://www.atlassian.com/git/tutorials/

Git: https://www.atlassian.com/git/tutorials/what-is-git

The following diagram has been provided to help you understand how Git interacts with your computer and Atlassian Bitbucket. The provided commands are only a few of the

many commands available in Git. You are encouraged to explore these further on your own.



What to do: You are supposed to complete the implementation of the following code and submit it to a Git repository.

IMPORTANT NOTE: This is group work and all 3 students must do their own code commit. You may not work by yourself on this assignment.

Please proceed with the following steps:

1. Every student must create a Bitbucket (Or GitHubt) account, and install Git on his/her computer. You can follow this tutorial to do so:

https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud

2. One member of the team must create a shared project in Bitbucket and give permission to all members of the team to access the project.
    a. Name your repository as "Project_First_Second_Third", where you replace "First", "Second" and "Third", with the first names of your teammates.
    b. Give Read and Write access to each member of your team using the email address that they provide you.



3. The second student of the team will create a java file (Converter.java) with the content shown below and will submit the file to the shared repository. This code has a main function and two other functions. The main function calls the other two conversion functions. The conversion functions convert Celsius to Fahrenheit and vice versa. The celsiusToFahrenheit function receives a Celsius value and converts it to Fahrenheit. The fahrenheitToCelsius function receives a Fahrenheit value and converts it to Celsius.
4. Next, each student will pull the shared java file, and will implement a function assigned to them in the code comments. For example, based on the comments, the third student must implement the celsiusToFahrenheit function.
5. After writing the code, each student must add his/her name to the list of the authors of the file (i.e. in front of @Author tag in the beginning of the file) and then push the modified code to the repository.

```
public class Converter {
        //Your names go here:

        /*
        * @Author:  Name of the first student
        *              Name of the second student
        *              Name of the third student
        *
        */

        private double celsiusToFahrenheit(double C){
           // TODO: The third student will implement this method
           return 0;
        }

        private double fahrenheitToCelsius(double F){
           // TODO: The second student will implement this method
          return 0;
        }

        public static void main(String[] args) {
          //TODO: The first student will implement this method.
          //     Call CelsiusToFahrenheit to convert 180 Celsius to Fahrenheit value.
    //    Call FahrenheitToCelsius to convert 250 Fahrenheit to Celsius value.
        }
}
```

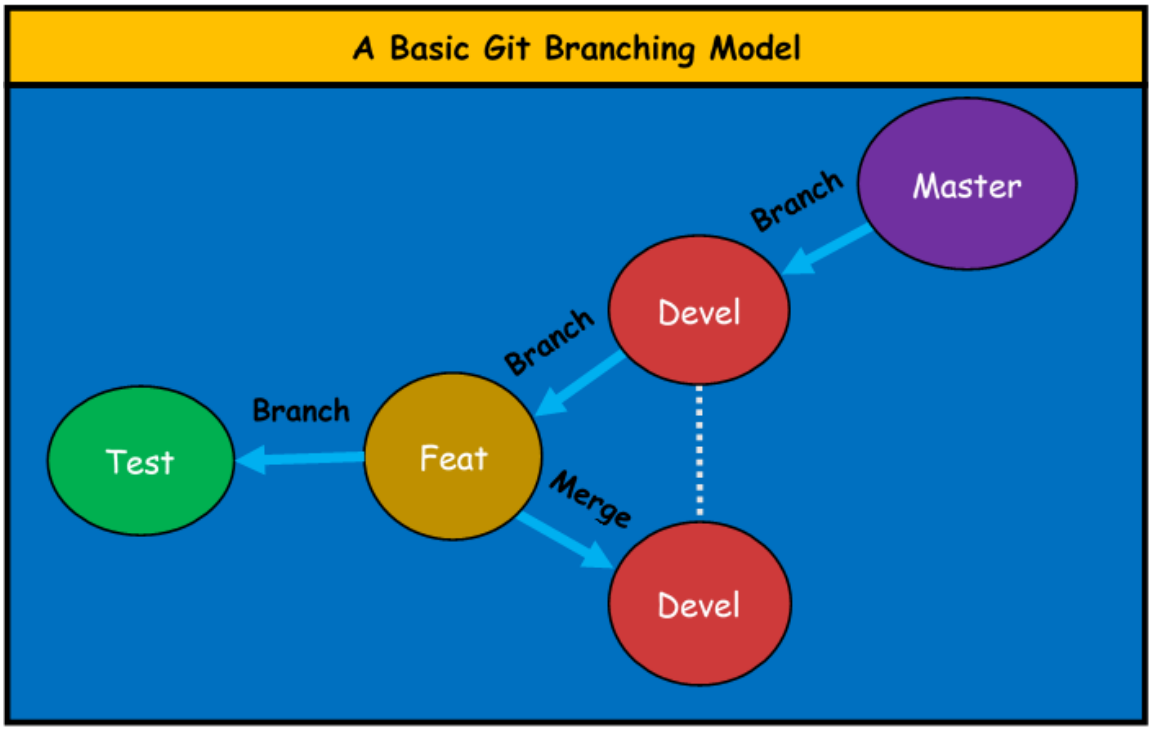
What to hand in: There is nothing to be submitted in this assignment. You need to finish the above steps and provide a demo to your TA as stated on the cover page of this lab.

Note 1: This assignment must be done in group. Individual work will not be accepted.

Note 2: Your TA must be able to see the changes made by all group members in the history of the repository. If for any reason the history of the project is deleted, the team members must re-do the exercise.


Exercise - 2: An Introduction to Branching

Suppose you are deploying a project, which you have placed in a Git repository. This code is in your `master` branch. After the project is deployed you realize there is features you want to add, but you do not want to pollute your deployed code with untested, or incomplete code while the new feature is developed. So, you decide to create a new branch from `master` called `develop`. Now the `develop` branch will have a copy of the code from `master`, and changing code in `master` or `develop` will not affect the other. The new feature can now be created in `develop` without worry. Once the feature is finished you realize you want to test it, and if these tests are successful, the feature will be merged into the `master` branch. However, you only want to merge the new feature's code and not the testing code. You decide to create a new branch from develop called `test` to include the new feature and your tests. Then you can easily merge the `develop` and `master` branches without any code from the `test` branch. This is an example of a simple Git branching model.



Branching is a way to keeps track of multiple versions of the same code which are independent of one another. The advantage of branching is that you can implement new features in one branch and the code in other branches will remain the same unless you "merge" that branch with it. Typically, the `master` branch will hold finished code that is functional and well tested while separate branches are used for developing and testing any new features. Once these new features are finished, they are typically merged with the `master` branch, or a `release` branch.


What to do: You are supposed to complete the implementation of the following code and submit it to a Git repository. This repository will be the same repository used in exercise 1, with the same group members.

Please proceed with the following steps:

1. The third student of the team will create a new branch in the repository called `devel` (a.k.a. develop) and will add a comment that says "This is the development branch" to `Converter.java`. Then the student will push the java file and `devel` branch to bitbucket.

2. The second student will create a new branch from `devel` called `feat` (a.k.a. feature). Then they will implement a new function called `kilometersToMiles`, which receives a Kilometers value and converts it to Miles. Then the code must be pushed to the `feat` branch.

3. The first student will create a new branch from `feat` called `test`. Then they will test the new function in the programs main method. This test should call the method `kilometersToMiles` with the value 30 Kilometers as an argument. This version of the code should be pushed to the `test` branch.

4. The third student will navigate to the `devel` branch and merge it with the `feat` branch. If done correctly, the code in `devel` will contain the `kilometersToMiles` method but not the testing code from the `test` branch, and the `master` branch will only contain the solution to Exercise 1. The source code in each branch can be compared by using the `diff` command in Git or by viewing the repository's bitbucket page.

Hint: If code is accidentally pushed to the wrong branch, then Git has several commands to view branch history and revert back to previous versions.

What to hand in: There is nothing to be submitted in this assignment. You need to finish the above steps and provide a demo to your TA as stated on the cover page of this lab.