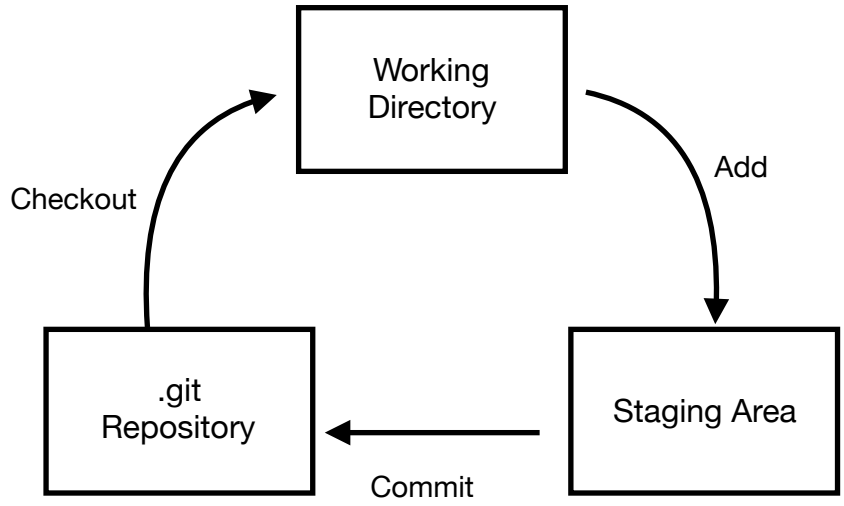
ELEC 377 Operating Systems
Git Overview

Github is a distributed source code control system that keeps track of multiple versions of your source code. Unlike other version control systems that store each version of the file as a delta (or difference) from previous files, git stores snapshots of the development directory. To save space, any files that are the same as the previous snapshot are not stored. Only changed files are stored. Git has a lot of capabilities, however, since most students have not used git, or any similar version control system, we will only be using a small subset of git for the class.

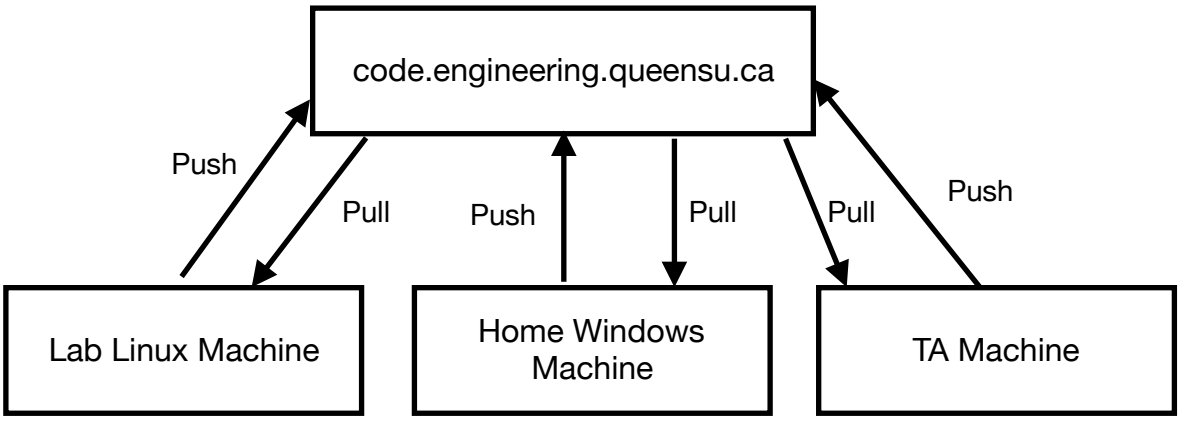
Git has the ability to manage multiple branches. For example, a developer may have released a version of the software (say 1.0). While working on the new version (say 2.0), the previous version may need an update because of a bug or security vulnerability. Git can track the two different branches, allowing an update to version 1.0 (now 1.0.1) while keeping the changes for the new version different. In this course we will only use the default branch, called the *master branch*. Thus commands like *checkout* which are used to switch branches will not be used.

On your local system, there are three areas that are part of git. These are the working directory, the staging area and the repository. The working directory contains the version of the files you are currently changing. The staging area and repository are in the hidden directory *.git*. Don't try to manually change anything in that directory. In the case of ELEC 377, the repository is a copy or *clone* of your repository on code.engineering.queensu.ca. This repository has the history of all files and the changes made to the files during your labs.

After you edit some of the files in the working directory, you may want to save the changed versions to the repository, but you may not want to save all of the files that have changed. You indicate which files are to be saved by adding them to the staging area (*git add filename*). In our case, we will usually add all of the changed files by using "*git add .*" Which adds all of the files in the current directory (the "." means current directory). However you can choose to add each file to the staging area individually with multiple *git add* commands. Once all the files are in the staging area then the command *git commit* will copy all of the files from the staging area to the repository. Commit commands must also contain a message (-m flag) which describe the changes. The following figure shows the lifecycle of the files. The *checkout* of the files is done as part of the *clone* command.



While this protects from losing a change to a file (all changes are in the repository), it doesn't protect from losing all of the file versions if the hard dive fails, or if you corrupt your virtual machine. Git supports distributed repositories. In the case of ELEC 377, there are at least two repositories. The one on code.engineering.queensu.ca, and the one on your local linux machine. If you choose, you can download the linux images to run in Virtual Box or VMWare player, or you can also clone your repository to a Windows machine as well. The markers will also have made a clone of your repository. In this case the repository on code.engineering.queensu.ca is called the *upstream* repository and the cloned copies are called downstream *repositories* as shown in the following diagram



When you commit, you increase the version number attached to the current branch.
Any changes to the repository must apply the version numbers in order. When there is only one local and one remote repository, this does not present a problem You make changes to the current version by committing changes to your current working directory, stage them (add), commit them (increments the current version in the local repository) and push them (send the new version to the upstream server. However when there are more than one downstream repositories things can get out of sync. For example you make a change on both the lab machine and another machine, or a change directly on the upstream repository, then both have a new version and they are not the same.

There are three common problems that may occur in ELEC 377.

Different Files
The changes may be to different files. For example, at the same time that you were checking in your documentation, the TA may have checked in the skeleton files for the next lab. Either you or the TA will have done a push first. The other one must do a pull before they can do a push. The pull will update the local repository, and the current local commit will be incremented to the next version and can be pushed. The repository *merges* the changes since different files are affected.

One or more Common Files.
If you have changed the same file on more than one repository, this is called a merge conflict. The two commits cannot be merged since the same file has been changed. In this case the second person has to fix the conflict. After the pull, the file in question will have both versions in it. Git will attempt to find the common lines. Lines that are different will be marked using three lines. The first line '<<<<<<<<HEAD' identifies the beginning of lines from the remote repository. The end of

these lines is singled by a line of equal signs '==========' followed the lines for the local version. The end of the local lines is marked with '>>>>>>>>>> branchname'. There may Be several instances of these differences in the file. You have to choose which version to keep and then add and commit the file again.

Changes between three repositories
The conflicts must be resolved by each repository separately.