**Additional Resources - Git and Github**

Vocabulary: Repositories, Commit, Branches, Pull, Push, Fetch, Clone, Fork, Revert, Pull Requests, Merge Conflicts, Git, GitHub, Version ControlAdditional Resources

[Github's Guide to Create a new repository (Links to an external site.)](https://guides.github.com/activities/hello-world/)

[Atlassian Guide on Git (Links to an external site.)](https://www.atlassian.com/git/tutorials/what-is-git)

[Github Student Developer Pack (Links to an external site.)](https://education.github.com/pack)

[Git vs. Github (Links to an external site.)](https://www.codefellows.org/blog/git-and-github-what-s-the-difference/)

**Lesson**

Git is a version control system for tracking changes in files, and being able to collaborate with others to work on the same files together. This lesson will teach students how to use Git and how it is used in order to collaborate with others. It is critical in a work environment to be competent with Git. It is the primary mode in which you will submit and receive code to craft into mobile applications.

**Repositories**

A repository is the home for your code. Changes that you make in code will be reflected in your repository for that code. All folders and files in your projects can be stored in a repository. There are many services online that you can utilize to save your repositories. For this class, and for many people in the programming community Github will be our service that we use. So, how do we start with a repository on GitHub? All you need to start is a name, but you also will have many options to include which we will cover.

Typically you keep the repository name to be the same as your project. So for a Journal App your repository could be Journal, or the name of your app.

You get the option to make the repository public or private. With a free account you will only be able to create a public and private repositories. A public repository means that anyone on the internet would be able to open and look at your code. A private repository only lets you and your collaborators access the repository. Students can qualify for Github for free! Check out the Student Developer Pack in Additional Resources.

In addition your repository can include a README. When possible you should create a README for all of your repositories. Consider this as an introduction to what your repository is all about.

The last two options you get are to add a .gitignore and a license. The Git Ignore is way to hide files in your repository. A good example is .DS*Store. That type of file type is a nuisance as it is created whenever you create new files and folders. Adding .DS*Store to your .gitignore is common practice. Licenses are there to protect you. For example, if you had a code base that other people are using. You will want to be acknowledged for your work, an Apache License will require anyone using your code to provide attribution to you.

There are millions of repositories on GitHub. The public repositories you can access and learn from. If you just want to open the project then you can download the ZIP. However, you have other options depending on what you want to do.

Your first option would be to Clone. Cloning a Repository gives you a copy of the code that is in the repository. Another option is to Fork. When you fork a repository you create a new repository that is identical to the original, but it is now on your Github account. Remember if you ever decide to use code from other people they probably have a license so you will need to attribute them in your project.

Commits and Branches

A commit is the bread and butter of Git. Think of committing as saving your code. These saves can be referenced and you can see the progress of the code.

In order to create a commit you need two things. A change in a file and a commit message.

While working on a project you are constantly changing code and writing new code. Anytime a file is created or adjusted it is added into a section called Unstaged Changes. When you are ready to commit you can move the files from Unstaged to Staged. When a file is in Staged Changes it is ready to be committed to the repository.

The second thing you need is a commit message. These messages should be descriptive of what is reflected in the staged changes files. For example if you just made a change in your project to fix a bug. Your commit message could say something like "Keyboard bug fixed"

**Branches**

By default you are given a branch called "master". Branches are a great resource for you to organize your code and to more easily collaborate with others on your project. Branches are essentially a place to hold commits. A good separation of commits are with features. You can start a new branch when you start working on a new feature.

It is good practice to have multiple branches. For an app that is in production it is common to have the master branch have the code that is live. A develop branch is commonly created to seperate what you are working on with what is in production. This enables developers to work in an environment where they do not need to worry about breaking code that is in production. Once code is tested on the develop branch it is merged into the master branch and put into production.

Another benefit to having branches is for collaboration. Later in the lesson you will learn about merge conflicts. When you and another person are working on the same project, you don't want to be changing the same lines of code. To do this it is best to create separate branches to work on your features.

How Git interacts with Github

Git and Github are not the same. Although they work hand in hand making each other better.

**Git**

So what is Git? Git is a version control system. It tracks versions of your projects and allows you to revert back in time to a previous commit. Git allows you to code away and not worry about breaking something. If something goes awry you know that you have a commit that you can go back to at anytime.

**Github**

What is Github? Github is a place for you to store your repositories. Instead of keeping everything local on your machine, Github gives you a place to store your code online. This allows for other people to see your code, or just a save haven for you to feel comfortable knowing your code is safe.

Pull, Push, Fetch

These are functions to help you have the right code on your computer. They help you interact with your GitHub Repository.

**Pull** is taking commits that are on the GitHub Repository and bringing them down onto your computer. For example, if Bill made a new commit and Jane wanted to see that commit she would need to *pull* down the code to see it on her computer.

**Push** is taking commits that are on your computer and *pushing* the commits to the GitHub repository. Until you *push*, your code is only stored on your computer. After a *push* then your commited code is both on your computer and on a GitHub Repository.

**Fetch** is telling your computer to go look at the repository in GitHub to see if there are any changes. If there are changes your computer will suggest that you *pull* down the changes.

**Pull Requests**

When you are working on a project with other people it could become a real disaster for everyone to be committing code to the same repository all the time. To solve issues with this we have *pull requests*. A pull request let's you commit your changes then allows a manager of the repository to either accept or deny your commits. These are commonly used for code reviews. A programmer will submit his code as a pull request then the manager will look over the code and determine whether it is ready to be merged in with the rest of the repository.

**Merge Conflicts**

A **Merge Conflict** is when you pull down code that someone else has changed and it interferes with code that you have changed. For example, if Bill changed line 24. Jane pulls down Bill's code and Jane had also changed line 24. This results in a Merge Conflict. It is always best to *pull* before you push. That way any merge conflicts that you may encounter are handled locally.

To solve a *Merge Conflict* you choose what lines of code you want to keep. Essentially going line by line determining who's code to take. You also have the option to always take Bill's code or always take Jane's.

**Conclusion**

Git is crucial for every developer. Committing often and having meaningful commit messages can really aide your development. Rarely will you work on a project by yourself. Being able to utilize Git to work with others will set you apart as a developer. One of the biggest headaches is working through Merge Conflicts and how to get the right code. Being able to master Git is critical for a successful developer career.