The use of software such as Git, Docker, automated testing, and continuous integration are key for the success of any company. Especially if they’re a tech company. This is apparent when you look at the productivity of any company that uses these software's. They complement teamwork and efficiency, without it being necessary to be “together”. This is key especially in this current COVID-19 environment. Git and Docker makes it possible for any number of people to work together remotely without distractions. Git is seamless in it’s approach to teamwork and productivity. It allows you to see your teammates work and branch off to create your own work or even edit their work. As shown below, due to Git’s many features and functions, teams are allowed to work together seamlessly.

**Repository:**

A Repository is the location where files are stored on Github. If a company wanted to collaborate on a project simultaneously, they could all connect to the Github repository and download it onto their own machines. This would allow them to access the main repository on their machine while being able to make changes and push them to the main repository.

**Clone:**

A clone is a duplicate of the repository so that edits can be made without interfering with the main repository. Although changes can be made without pushing changes to the main repository, using ***git clone*** will be able to show multiple changes that people make and compare how the repository would act without major changes on the main.

**Fork**

***Git fork*** is similar to git clone, but it has one major distinction. While git clone copies the repository onto the local machine, git fork makes a duplicate onto the repository on Github. This allows changes to be made to the forked repository without having to push the changes first, and the changes done can be merged with the original repository or whatever designated location.

**Branch**

A git branch is an alteration to the path you are currently working on which is not set to be integrated with that path. If someone wanted to make a change on a project but wanted it to be worked on separate from the main path, a branch could be made and the person could continue working on it and making commits while other branches were worked on at the same time. When they wanted to commit it to a different path, they could merge the two and combine what they have done.

**Commit**

***Git commit***is publishing the changes you made into the project repository. This can be done on remote repositories or on the Github repository, but if it is done on the remote repository the changes must be pushed before they would show up on Github.

**Merge**

**A Git Merge** is the act of combining the contents of one branch integrating it with the contents of a target branch. This is ideal for making two independent separate branches into one single branch, combining multiple commits into one unified branch. A merge is done after two people are done working separately and want to combine their branches.

**Checkout**

**Git Checkout** is the command used to update the files in the branch and tell Git to store all the new commits to the new branch. For example, when one is done adding a commit to a branch, they would enter git checkout in order to officially make it part of the branch.

**Push**

**A Git Push** is the command used to upload items from the local repository into a remote repository. Git Push is essential for remote teamwork. It allows for work on separate repositories to match each other. Git push will be used by anyone who would like to edit someone else’s repository, or help them with something.

**Pull**

**A Git Pull** is the opposite command of a Git Push. Instead of uploading from a local repository to a remote repository, Git Pull downloads content from a remote repository and updates the local repository and matches the content with each other. A Git Pull is essential for any team, without it, all remote work would not be added to repositories.

**Status**

**A Git Status** is the command used to understand what has happened in the repository. It’s used to know what changes were made, what was added and removed, etc. It checks the state of each command.

**Master Branch**

A master branch in Git is the main branch as to which every commit is started from. It is the default branch given when starting a commit. All subsequent branches are made from the master.

**How to set up Git and work collaboratively with more than one person:**

**Step 1. Create a Repository**

A repository is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs. It is recommended to include a README, or a file with information about your project. GitHub makes it easy to add one at the same time you create your new repository. It also offers other common options such as a license file.

For example, “IS601\_Project\_1” is the repository we created for mini project. This repository can be a place to store ideas, resources, or even share and discuss things with others.

**To create a new repository**

In the upper right corner, next to your avatar or identicon, click and then select New repository.

Name the repository “IS601\_Project\_1”.

Write a short description.

Select Initialize this repository with a README.

Click Create repository.

**Step 2. Create a Branch**

Branching is the way to work on different versions of a repository at one time.

By default, repository has one branch named “main” which is the definitive branch. We use branches to experiment and make edits before committing them to “main” branch.

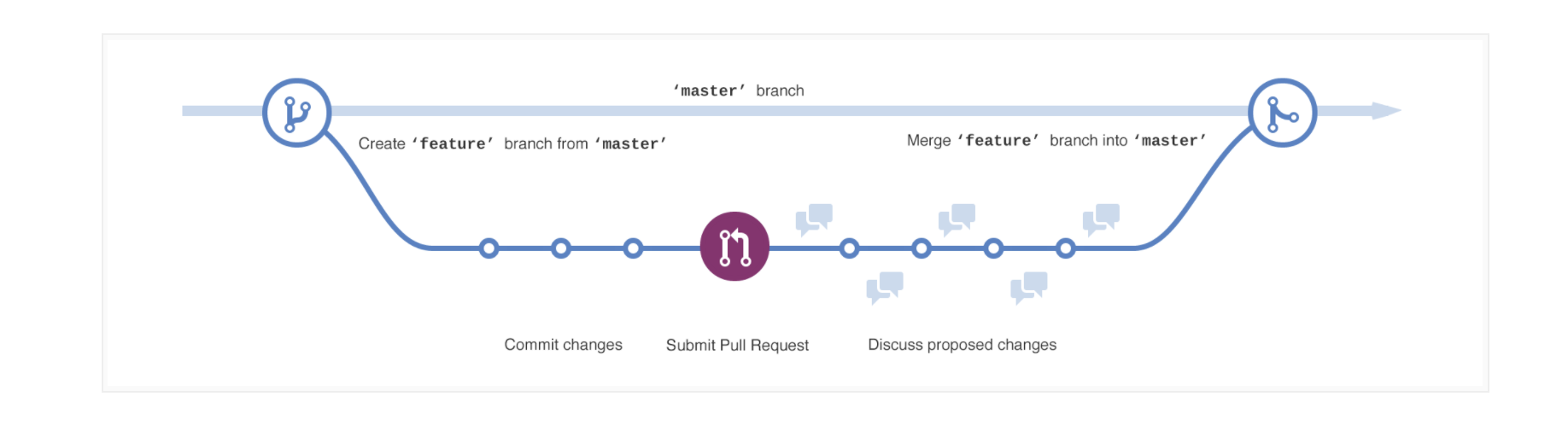
When you create a branch off the “main” branch, we are making a copy, or snapshot, of main as it was at that point in time. If someone else made changes to the “main” branch while you were working on your branch, you could pull in those updates.

Below Diagram shows:

The main branch

A new branch called “feature” (because we are doing ‘feature work’ on this branch)

The journey that “feature” takes before it’s merged into main



We can save different versions of a file. Something like:

Basic\_Git\_Commands.doc

Basic\_Git\_Commands edited.doc

Updated Basic\_Git\_Commands edited.doc

Branches accomplish similar goals in GitHub repositories.

In GitHub, developers, writers, and designers can use branches for keeping bug fixes and feature work separate from main (production) branch. When a change is ready, we can merge the branch into main.

**To create a new branch:**

1. Go to your new repository “IS601\_Project\_1”.
2. Click the drop down at the top of the file list that says **branch: main**.
3. Type a branch name, “feature” into the new branch text box.
4. Select the blue **Create branch** box or hit “Enter” on your keyboard.

Now we have two branches, main and feature. They look exactly the same, but not for long! Next, we will add our changes to the new branch.

**Step 3. Make and commit changes**

Now, we are on the code view for “feature” branch, which is a copy of main and we can make some edits.

On GitHub, saved changes are called commits. Each commit has an associated commit message, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you have done and why.

**Make and commit changes**

Click the README.md file.

Click the pencil icon in the upper right corner of the file view to edit.

In the editor, write a bit about your project and about teammates.

Write a commit message that describes your changes.

Click Commit changes button.

These changes will be made to just the README file on your feature branch, so now this branch contains content that is different from main.