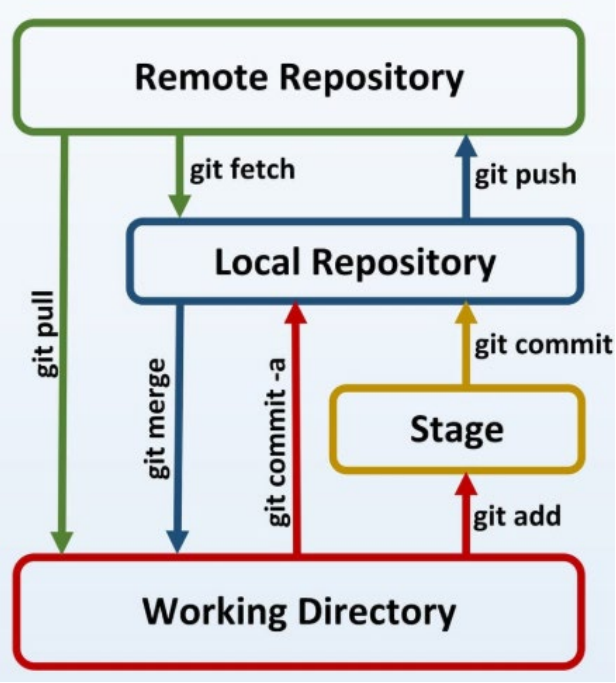
SOFTWARE ENGINEERING FUNDAMENTALS

TUTORIAL 1: BASIC GIT

In this session:

- You will learn the basics of Git and GitHub
- You will practice basic commands for Git
- You will start preparing for the first Individual Assignment

Basic parts of a Git repository:



PART A: Basic Git Exercise

1. Create a folder in your computer. This will be your working directory.
2. In the explorer, do Right Click -> Git Bash Here
3. Type the first command *git init*. This creates a new repository, and tells git that this is were you are going to start working. This is your local repository.
    a. You won't be able to see any changes folder. You need to toggle the "hidden folders" options to actually see what this does. Now your project will have a ".git" folder as well.
4. Right click, and create a text file named *mytextfile.*txt. Write something inside the text file. Save and close the file.
5. Return to git bash. Now write: *git add mytextfile.txt* This moves your file to the "staging area" indicating git that when you "commit" (i.e. save) your changes, whatever you did to this file, should be included.
6. Now another command: *git commit -m "adding my first file"*. This will actually save your changes from the staging area, to the local repository.

7. Create a new repository on GitHub by going to the + sign on the top right.
8. Put a name on your repository, and create! This creates the remote repository for your project.
9. You will see that the page now has different titles. Find the one that says: "…or push an existing repository from the command line". Copy these commands, and paste on GitBash.
10. Refresh the repository website. Your code is on the web!

On this part, you started from the working directory, and moved to the remote repository.

PART B: GitHub Classroom + Cloning + Practice

1. Go to the Git URL listed in Canvas
2. Accept the assignment and wait until your repository is created.
3. This repository is individual for you and is full of “katas” (mini exercises) that you can do to practice your git skills. They will be needed!
4. Once the repo is created, go to a folder explorer in your computer.
5. Create a new folder named “gitKatas”. Right click, and open git bash. Type: *git clone* URL (replace URL for the URL of your newly created repository). Now wait, your remote repository downloaded everything into your local repository, and it is visible in the working directory.

Use the cheatsheet to complete the following katas:

1. Basic Commits
2. Basic Staging
3. Basic Branch
4. Fast Forward Merge
5. 3-Way Merge
6. Merge Mergesort -> psst! The assignment may ask something like this.

BASIC COMMANDS
Check your cheat sheet for more information!

Clone — Clone makes a copy of a repo. Copying a repo is desired in case someone else needs the code. But more often than not devs use it to make an offline, locally stored copy of the Github repo. Working directly on the remote copy is not desirable.

Commit — Once a developer has cloned the code on her local machine, she can start writing code. Again, new code should go in a new branch. Once done, she can commit the changes. Commit saves the changes to local repo.

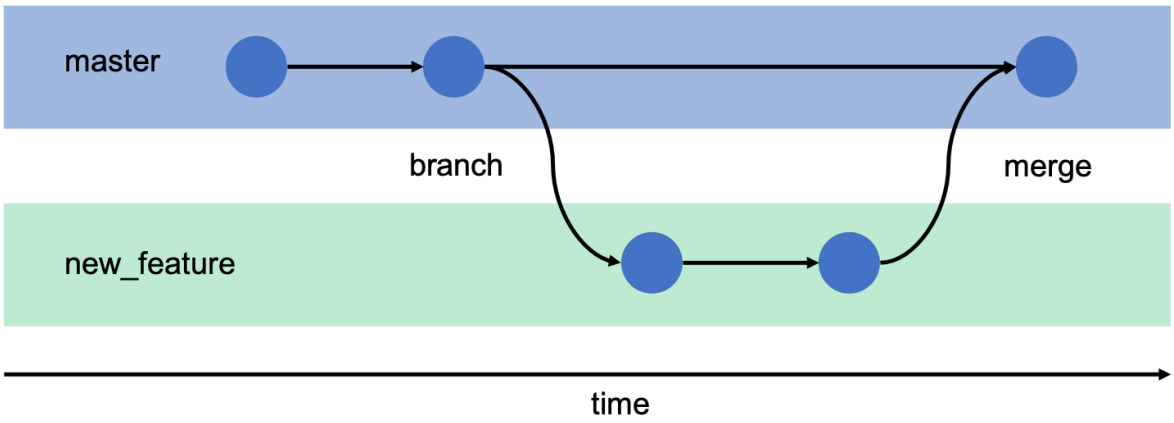
Push — While commit saves the changes in the local, push transfers those changes from local to the remote stored at Github.

Pull Request — Pull Request or simply PR is at the heart of Github collaboration. Supposed you made changes to your new branch. Pushed it to remote Github repo so everyone else could see but no-one is aware of your work as yet. Everyone is busy in their own work. Pull Request is a way to ask other devs to review your code. A PR shows the differences between two branches in green (additions) and red (subtractions).

Merge — Once someone has reviewed and approved a developer's changes, it's time for merge. Merge command merges new code with the master branch.

BRANCHING

Branching is a feature available in most modern version control systems.



Git branches are effectively a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes. This makes it harder for unstable code to get merged into the main code base, and it gives you the chance to clean up your future's history before merging it into the main branch.

.GITIGNORE

There are some files, folders and things that you don't want to push in your repo. If you write it in .gitignore, the command "git add" will always ignore those files.

Each line in a gitignore file specifies a pattern. When deciding whether to ignore a path, Git normally checks gitignore patterns from multiple sources, with the following order of precedence, from highest to lowest (within one level of precedence, the last matching pattern decides the outcome).

Find the patterns: https://git-scm.com/docs/gitignore

There may be an HD exclusive question on the Week 3 assignment regarding .gitignore! This may give you some hints, though: https://stackoverflow.com/questions/1274057/how-to-make-git-forget-about-a-file-that-was-tracked-but-is-now-in-gitignore