Git Notes

- Git is a control software. It removes the need to copy files to and from the class share and your “H” drive as you collaborate on a project.
- Is like using a camera to take a snapshot of your files at a specific point in time so you can go back to the moment the code was written, before terrible things happen.
- Purpose is to modify/change/break/improve your code. Basically so you’re confident that you won’t ruin your work too badly because you created save points along the way.
- It’s a collaborative tool, that allows different people to work on all the parts of a project at the same time.
- Protects yourself and others from yourself and others.

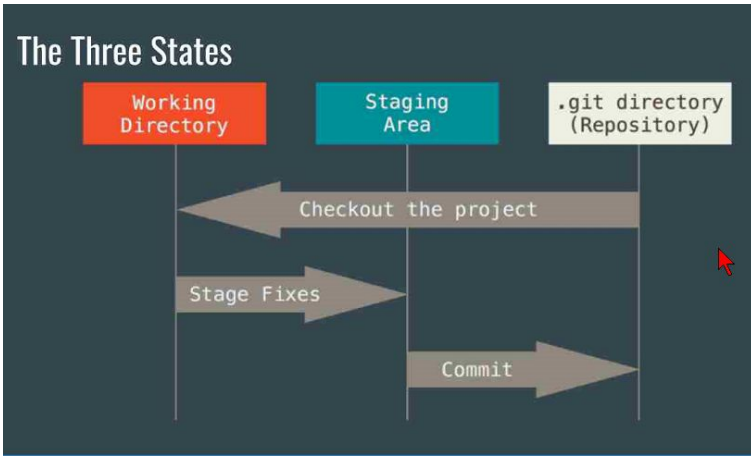
The Local Workflow

- Starting
    1. File Explorer, and create a folder named “practice” in “H” drive.
    2. Type “cmd” in the address bar (Basically open a cmd in folder you want to have a git in)
    3. Open a command prompt, it should open to H:\practice >
    4. Tell git to watch this folder by initializing it: **git init**

Git init creates a **repository** in the folder you ran the command on. Often shortened to **repo**, this is a hidden location where file **checkpoints will be stored**.

Three main “states” so that git is tracking your working directory.

- **Modified** - files that are new or have changes not yet saved by Git
- **Staged** - the current version of a file, tagged to be included in the next commit.
- **Committed** - files that are safely stored by Git



Task - Open Brackets and save an index.html file in your practice folder. Add the basic HTML structure, but nothing else and save the file.

Using Git

- Switch to your **cmd window** and check the status of files with: **git status**.
- Git **tracks changes** to your files.
- **Red** indicates that the files marked hasn't been saved.

Checkpoint

- To save, tell Git to **stage** our file. Do this by adding the file in cmd: **git add index.html**
- Check **git status** and the previously **red** file is now **green**.
- **Git add** neatly packs a **copy** of specific file changes into a box, ready to be stored indefinitely.

Storing a box

- We **commit** the box to storage and note what is contains.
- **git commit -m "description goes here"**
- Until we **commit** our work, there is **no checkpoint** to save us from errors.
- The commit description, should explain what changed.
- **Git commit** physically moves the box of **copies** into long-term storage, so describe what's in the box just in case you need it. It **does not** move or remove files in your working directory.

Checking status again

- "Nothing to commit, working tree clean" - Nothing is in progress anymore. Its stored.

Task - Change **title** of HTML to your name **Save** the file.

1. You'll see: **modified: index.html**.
2. **Add** and **commit** the change:
3.
    - **git add index.html**
    - **git commit -m "updated title"**
4. **Status** should be **clean**
5. **Status, Add, Commit, Status**.
6. **git log**
    - Returns a list of all the commits (boxes) you've saved.
    - Gibberish is the unique name of the box.
    - Author and Date is visible, with descriptions that you've made.
7. Add **p** tag saying **something nice** about the person next to you.
8. Add an "**h2**" that contains what you consider the "big idea" of the lesson was.
9. Write a complete "**p**" that summarizes today's lesson in your own words and describes the 3 states a file may be in and what they mean.
10. On a scale of 1 (I'm lost) - 4 (I got this) how would you rate your understanding of git?
11. Add, Commit, Log your changes.

12. Turn in your html file and screenshot of your last git log on classroom.

Continued notes

Remote repository - copy of our project that's stored "in the cloud" aka just in another computer. It's where we **backup** our work and share it with others. Accessible **anywhere** that there's an internet connection.

Task - Create an account @ github.com

Linking our local repository:

1. Locate your project in File Explorer.
2. Initialize your local repository
3. Add and Commit your existing files
    a. **git add .** (shortcut to add all files)
    b. **git commit -m "initial commit"**.

Linking our remote repository:

1. Tell git about the remote repository (once per computer):
    a. **git remote add origin https://<YOUR URL>.git**
2. **Push** your changes to the remote server (**as needed**):
    a. **git push -u origin master**
    b. After the first push it can simply be:
        i. **git push**

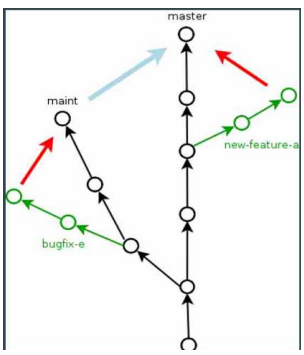
Git push tells git to upload all your changes to the server. It **doesn't** need to be done after every commit because it will upload ALL commits since the last push.

Working with branches

Branches are pretty much what they sound like. Smaller bits extending from a tree trunk. They represent different versions of our code.

Allows us to work on code fixes and feature without breaking what we have already working.

Fixes and new features should **always** start on a branch that is NOT the master branch.



Master branch is the "trunk" of the code tree, and should only contain **clean code ready for deployment**.

Git branch <name> tells git to maintain a new copy of our code with the given name. Git branch by itself will list the branches available and display an asterisk (*) next to the one we are currently working on.

Git checkout <branch> tells git to switch our working folder to the branch name specified.

Working with branches:

We're going to add media queries to our flexbox page.

1. Create new branch so we don't break our working code:
    a. **git branch mobile**
2. **Important!** Switch to our new branch:
    a. **git checkout mobile**

Task

1. Open Brackets and ad a MQ line to flex.css:
    a. @media screen and (max-width:480px){}
    b. Save the file
2. **Git status** to see changes
3. **Commit** them.
4. **Push** them and check the **github** page.'
    a. Its gonna give a message to fix it, copy it put it.
5. **Checkout master** branch
    a. **git checkout master**
        i. Did your flex.css file change? How?
        ii. What happens when you switch back to the **mobile branch**?

So we learn that when you **checkout** to the different branch (**master**), the file that you uploaded in the **mobile** branch switches to what you previously made. There are now two independent files.

Merging branches:

1. **Checkout master** branch
2. Look at your flex.css file in brackets.
3. Merge the mobile branch into the master branch with:
    a. **git merge mobile**
4. Check your flex.css file again now.

So now at this point, Whatever what was in **mobile** is now in the **master** branch.
Git merge <branch> combines the file changes in branch we name into our current working branch.

Tf is a merge conflict.
Merge conflict, is when a **file has changed in both of the branches** you are trying to combine and git can't automatically determine what you want to keep.

Basically **git is asking for help** because it is confused.

Merge conflicts:

1. Create and checkout a new branch named tablet:
    a. Git branch tablet
    b. Git checkout tablet
2. Create tablet size MQ in flex.css:
    a. @media screen and (max-width:1024px){}
    b. Note the line number you created this on.
3. Attempt to merge

4. Git **tagged** the conflict:
    <<<<<<< HEAD refers to the active branch.
    ======= separates the conflicting changes in the branches
    >>>>>>> mobile shows the branch the merge is coming from.

To resolve:

1. **Remove the tagging** and **keep the code** that we need.
2. **Save** the file, **add and commit** the changes to resolve the conflict. (keeping 1024px)

<u>Questions</u>

- Basically, this can allow multiple people to be working on the same file if they dedicate the line space area and have different branches for types of code to implement.
- I feel like I'm a 3. I think I'm a bit confused on exactly what is the differences between git commit, and git push. But by the time you read this, I'll probably have it answered.
- Not really.