GIT BRANCH (Revisiting Rock-paper-scissors)

With branching (in Git) you can make your changes without having to worry about breaking what you have now.

When you make commits on a specific branch, those changes only exist on that branch. This means that you can keep your main branch as a place for only finished features that you know are working properly, and add each feature to your project using dedicated branches which we call feature branches.

A. Using Branches

`git branch <branch_name>` : make new branches

`git checkout <branch_name>` : change position to your new branch

`git checkout -b <branch_name>` : create a new branch and change to it in a single command(sekaligus)

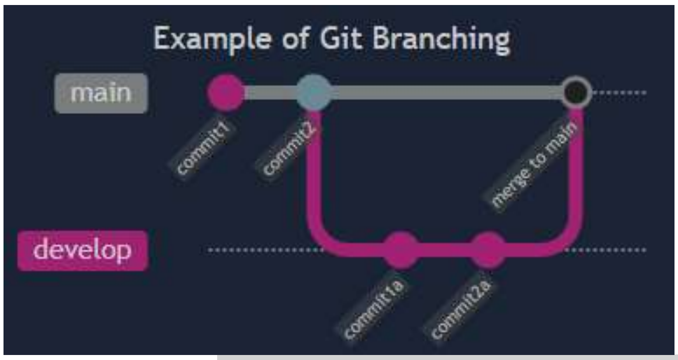
`git branch` : see all of your current branches.

The branch that you're currently on will be indicated with an (*)asterisk.

`git push origin <branch_name>` : to push in <branch_name> branch.

Once you are done working on your feature branch and are ready to bring the commits that you've made on it to your main branch, you will need to perform what is known as a 'merge'.

Merges are done by using the command `git merge <branch_name>` which will take the changes you've committed in 'branch_name' and add them to the branch that you're currently on. You can see an example of a 'develop' branch being created, committed to, and then merged to main in the diagram below.



Sometimes, the same lines in a file will have been changed by two different branches. When this happens, you will have a merge conflict when you try and merge those branches together. In order to finish merging the branches you will have to first resolve the conflict, which will be covered in a future lesson.

`git branch -d <branch_name>` : delete a branch (if the branch has already been merged into 'main')

`git branch -D <branch_name>` : if it hasn't.

You will usually want to delete branches when you're done with them, otherwise they can pile up and make it more difficult to find the branch you're looking for when you need it.

B. Sharing Code

Another great use case for branches is to share code with others that you might not want to commit to your main branch (or feature branch) at all.

For example: if you have a bug in a new feature you're working on that you can't figure out, and it causes your code to break, you don't want to commit that broken code and have it in your project's "permanent record". You could instead create a new temporary branch, switch to it and commit your code to this new branch. If you then push this new temporary branch to GitHub you can share it with others that may be able to help solve your problem.