**Version Control**

ALL PLAGIARISED

Version control is a system that records changes that are made to a file or a set of files over time.  If you’ve ever used the “Track changes” feature in Microsoft Word, you have seen a rudimentary type of version control, in which the changes to a file are tracked, and you can either choose to keep those edits or revert to the original format. the version control software keeps track of who, when, and why those specific changes were made. Version control allows multiple people to work on the same file and then helps merge all of the versions of the file and all of their edits into one cohesive file.

**Git**

Git is a free and open-source version control system. One of the main benefits of Git is that it keeps a local copy of your work and revisions, which you can then edit offline, and then once you return to internet service, you can sync your copy of the work, with all of your new edits and tracked changes to the main repository online. Additionally, since all collaborators on a project have their own local copy of the code, everybody can simultaneously work on their own parts of the code, without disturbing that common repository.

**GitHub**

GitHub is an online interface for Git. Git is software used locally on your computer to record changes. GitHub is a host for your files and the records of the changes made. Think of GitHub as being similar to DropBox.

**Version Control Vocabulary**

**Repository:** Equivalent to the project’s folder/directory - all of your version controlled files (and the recorded changes) are located in a repository. This is often shortened to **repo**. Repositories are what are hosted on GitHub.

**Commit:** To commit is to save your edits and the changes made. A commit is like a snapshot of your files: Git compares the previous version of all of your files in the repo to the current version and identifies those that have changed since then.

If you find a mistake, you revert your files to a previous *commit.* If you want to see what has changed in a file over time, you compare the *commits* and look at the messages to see why and who. Each commit should only address a single issue. This way if you need to identify when you changed a certain line of code, there is only one place to look to identify the change and you can easily see how to revert the code. making sure you write informative messages on each commit is a helpful habit to get into.

A picture containing diagram

Description automatically generated

**Push:** Updating the repository with your edits. Since Git involves making changes locally, you need to be able to share your changes with the common, online repository. Pushing is sending those committed changes to that repository, so now everybody has access to your edits.

Additionally, don’t horde your edited files - once you have committed your files (and written that helpful message!), you should push those changes to the common repository. If you are done editing a section of code and are planning on moving on to an unrelated problem, you need to share that edit with your collaborators!

**Pull:** Updating your local version of the repository to the current version, since others may have edited in the meanwhile. The files you have locally on *your* computer may be outdated, so you pull to check if you are up to date with the main repository.

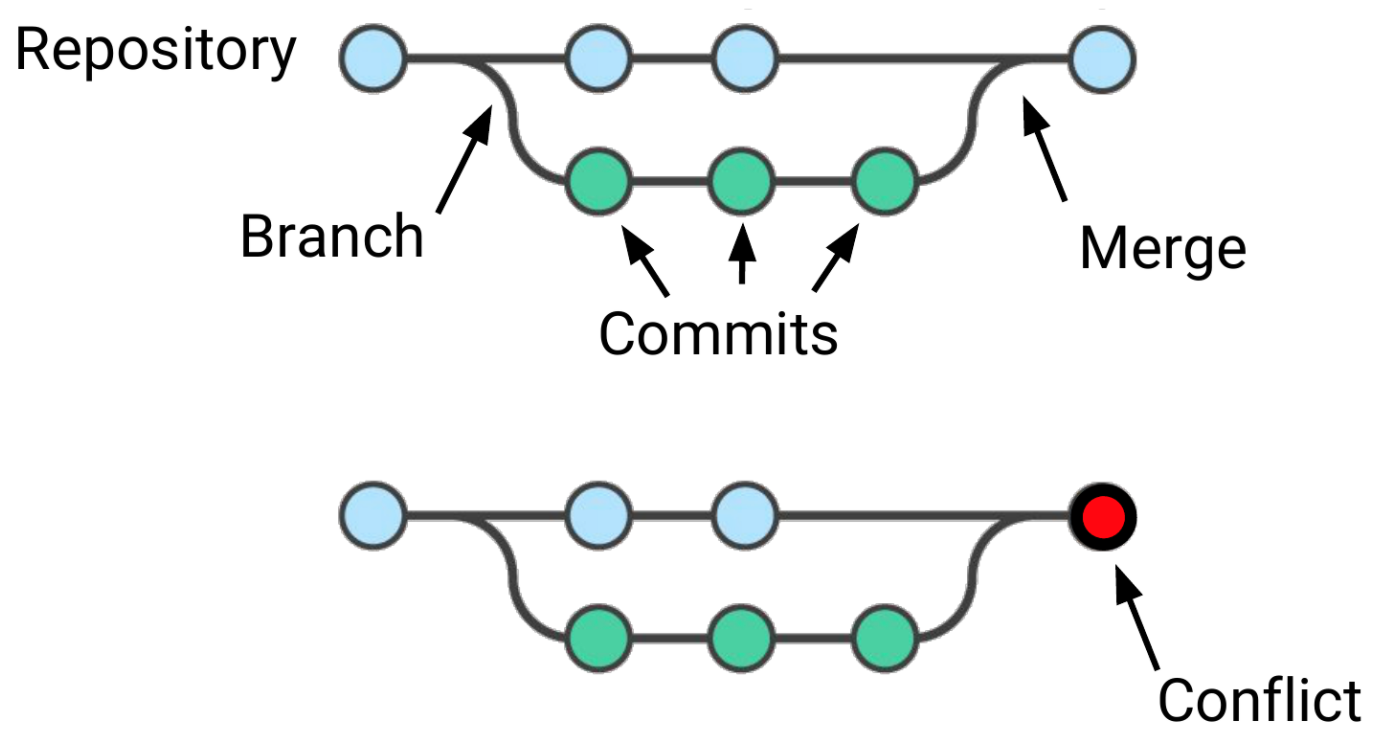
Frequently check that you are up to date with the current repo by frequently pulling.

**Staging:** The act of preparing a file for a commit. For example, if since your last commit you have edited three files for completely different reasons, you don’t want to commit all of the changes in one go; your message on why you are making the commit and what has changed will be complicated since three files have been changed for different reasons. So instead, you can stage just one of the files and prepare it for committing. Once you’ve committed that file, you can stage the second file and commit it. And so on. Staging allows you to separate out file changes into separate commits.

**Branch:** When the same file has two simultaneous copies. When you are working locally and editing a file, you have created a branch where your edits are not shared with the main repository (yet) - so there are two versions of the file: the version that everybody has access to on the repository and your local edited version of the file. Until you push your changes and merge them back into the main repository, you are working on a branch.

**Merge:** Independent edits of the same file are incorporated into a single, unified file. Independent edits are identified by Git and are brought together into a single file, with both sets of edits incorporated. But, you can see a potential problem here - if both people made an edit to the same sentence that precludes one of the edits from being possible, we have a problem! Git recognizes this disparity (**conflict**) and asks for user assistance in picking which edit to keep.

**Conflict:** When multiple people make changes to the same file and Git is unable to merge the edits. You are presented with the option to manually try and merge the edits or to keep one edit over the other.



**Clone:** Making a copy of an existing Git repository. If you have just been brought on to a project that has been tracked with version control, you would clone the repository to get access to and create a local version of all of the repository’s files and all of the tracked changes.

**Fork:** A personal copy of a repository that you have taken from another person. If somebody is working on a cool project and you want to play around with it, you can fork their repository and then when you make changes, the edits are logged on your repository, not theirs.