Version control system

A version control system, or VCS, tracks the history of changes as people and teams collaborate on projects together. As developers make changes to the project, any earlier version of the project can be recovered at any time.

Developers can review project history to find out:

* Which changes were made?
* Who made the changes?
* When were the changes made?
* Why were changes needed?

For me, it would be like a personal diary but for programming, digital and share with the team.

The extreme importance of knowing how to use a VCS, but in this case we are learning the most popular VCS that is GIt, ( a creation of the Linux `s creator)

* Git lets developers see the entire timeline of their changes, decisions, and progression of any project in one place. From the moment they access the history of a project, the developer has all the context they need to understand it and start contributing.
* Developers work in every time zone. With a DVCS like Git, collaboration can happen any time while maintaining source code integrity. Using branches, developers can safely propose changes to production code.
* Businesses using Git can break down communication barriers between teams and keep them focused on doing their best work. Plus, Git makes it possible to align experts across a business to collaborate on major projects.

If GIT would not exist, programmers should be sending copies of their codes all the time and the difficulty of tracking progress or even implementing changes to the main code would be really big, and that is why a VSC is so important to work with big programs on big teams of programmers.

Last semester I remember sending codes with screenshots by WhatsApp to see the code and correct or give an opinion upon the code of partners… Imagine trying to create a program working all together!

Something important to understand is that Git and Github are not the same… Git is the VSC that we use locally on our computer and it is essential we learn how it works and to learn in a terminal line way, that is by using commands on a terminal like GIT Bash that is not graphical.

To work with Git to initiate a project, first we have to configure our GIT in the terminal giving the username and email (Git config user.name “username” and git config –global user.email email) and we use the command –global to configure our git in a global way in our system

Then after configuring some other important aspects we need to create a repository, and let us think the repository not much as a folder or directory but the project itself containing not only code but different types of files and folders… the repository has the main and the branches. The branches are important to do not change the main repository until to be sure our code are ok to added or mixed into the main of the repository…. The basic command to do that are:

* git init initializes a brand new Git repository and begins tracking an existing directory. It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.
* git clone creates a local copy of a project that already exists remotely. The clone includes all the project's files, history, and branches.
* git add stages a change. Git tracks changes to a developer's codebase, but it's necessary to stage and take a snapshot of the changes to include them in the project's history. This command performs staging, the first part of that two-step process. Any changes that are staged will become a part of the next snapshot and a part of the project's history. Staging and committing separately gives developers complete control over the history of their project without changing how they code and work.
* git commit saves the snapshot to the project history and completes the change-tracking process. In short, a commit functions like taking a photo. Anything that's been staged with git add will become a part of the snapshot with git commit.
* git status shows the status of changes as untracked, modified, or staged.
* git branch shows the branches being worked on locally.
* git merge merges lines of development together. This command is typically used to combine changes made on two distinct branches. For example, a developer would merge when they want to combine changes from a feature branch into the main branch for deployment.
* git pull updates the local line of development with updates from its remote counterpart. Developers use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.
* git push updates the remote repository with any commits made locally to a branch.

To finish with this basic mastery of VSC, the GitHub is the program to share repositories between programmers and work together with the same repository to create a project as a team.