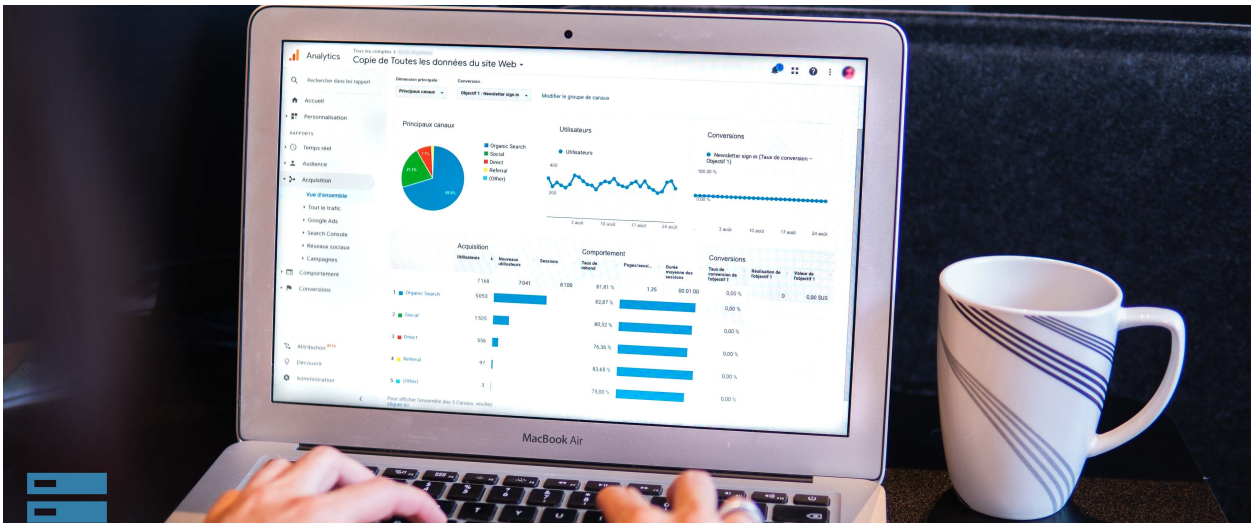


# Labs

Notes On Labs

▼ **Lab 1:**

- Git is a distributed version control system (DVCS)
- Working directory is the one version of project.
- when you add file you make it in Staged Area.
- when you modify on file you make it in Modified Area;
- when you commit file you make it in Unmodified Area.

▼ **Commands:**

1. **git init** ⇒ make a new repo in git.
2. **git status** ⇒ gives you the status of file .
3. **git add “file name”** ⇒ move file to staged area.
4. **git commit -m “your message”** ⇒ commit file on repo.
5. **git restore —staged “file name”** ⇒ restore file from staged area to modified area.
6. **git restore “file name”** ⇒ Undo the edit to the file.

7. **git reset —hard “file name”** ⇒ reset to last version of file from commit area.
8. **git checkout “hash”** ⇒ move file to the version of given hash.
9. **git checkout “branch”** ⇒ restore to last version that the master have.
10. **git log —all —decorate —oneline —graph** ⇒ to view all the previous commits.

▼ **Lab 2/Lab3:**

- **.gitignore** ⇒ plain text file you can put in it the files that not be tracked.
- you should commit the ignore file to git repo.
- **branch** ⇒ you can make it if you want to work in area and another one wants to work on the same project and you do not allow him edit on your code .
- When the HEAD points to a branch, it is called a **symbolic pointer.**
- When it points to a specific commit - and not a branch- it’s called a **detached HEAD**.
- You can’t delete the checked out branch.
- **A change Vs no-change, the change will win!**

▼ **Commands:**

1. **git branch “branch name”** ⇒ to create new branch.
2. **git branch** ⇒ list the branches you have.
3. **git log —all —decorate —oneline —graph** ⇒ to view git log history with more details.
4. **git checkout “branch name”** ⇒ to switch to another branch.
5. **git merge** ⇒ merge between two branches.
6. If there is **a direct path** from master to dev, Git will perform **a fast-forward** merge.
7. • As there’s no direct path from the master to math-feature, merge is done **recursively (3-way merge)**

8. **git branch -d “branch name”** ⇒ delete branch if this branch not be merged.
9. **git branch -D “branch name”** ⇒ delete branch forced even if this branch be merged .
10. **git rebase “base branch”** ⇒ take the **commits** in your checked out branch and apply them to the head of the passed branch.

▼ **Lab4:**

- **Access Control** : You can control who has access on your repositories.
- **Issues Tracking** : GitHub offers a built-in issue tracking system to manage and prioritize tasks, bugs, and feature requests.
- **Continuous Integration/Continuous Deployment (CI/CD)**: GitHub Actions can be used to set up CI/CD pipelines to automate the building and deployment of your software.
- **Web Hosting**: You can use GitHub Pages to host static websites directly from your GitHub repositories.
- **Creating a Pull request**: • A pull request (PR) is a request made by the owner or a contributor to merge their changes from a feature branch into another branch also called the **base branch.**
- **Merge Commit** : Creates a new merge commit, preserving detailed version history.
- **Rebase and Merge** : Replays the commits from the pull request on top of the target branch's latest state, for a more linear history.
- **Squash and Merge** : Condenses pull request commits into one, for a cleaner history.
-

▼ **Commands:**

1. **git clone “repo Url”** ⇒ To get a copy of a repository available on a hosting service.

2. **git remote add origin “repo Url”** ⇒ If you already have a local git repository, you can link it to your remote repository.
3. **git push** ⇒ push your commits or your new branches or any your updates from your local to your remote repo. **Simply ⇒ git push make remote to see your local.**
4. **git pull** = git fetch + git merge
5. **git pull** ⇒ this will look up what remote branch your current branch is tracking, fetch from that branch and then try to merge it in your current branch. **Simply ⇒ git pull make local to see your remote.**
6. **git push —set-upstream “origin branch name”** ⇒ • When you push a new branch that doesn't exist on the remote repository, you need to specify what the name of the corresponding remote branch will be using the following command.