Github QS:

[SOME CMDS AND CONCEPTS]

git init # Initialize a local repository
git clone <repo_url> # Clone a remote repository to local
git status # Check changes, staged and unstaged
git log # View commit history
git log --oneline # Compact view

```
C:\Users\ahmad\Downloads\quiz\ChatAURA>git log
commit 30cb4a3d43b1e31734097eedbe72ea5d78ab3bb4 (HEAD -> main, origin/main, origin/HEAD)
Author: دمحا دمحم <muhammad.ahmad.github@gmail.com>
Date:   Sun Dec 14 23:24:14 2025 +0500

    feat: pusheddd

commit 39c2b6a0f3f36b1053c524645c391cedbb262ac0
Author: دمحا دمحم <muhammad.ahmad.github@gmail.com>
Date:   Sat Oct 4 22:01:24 2025 +0500

    Initial commit

C:\Users\ahmad\Downloads\quiz\ChatAURA>git log --oneline
30cb4a3 (HEAD -> main, origin/main, origin/HEAD) feat: pusheddd
39c2b6a Initial commit
```

git diff # See unstaged changes
git diff --staged # See staged changes
git add <file> # Stage specific file
git add . # Stage all changes
git commit -m "message" # Commit staged changes

git branch # List branches
git branch <name> # Create a new branch
git checkout <branch> # Switch branch
git checkout -b <name> # Create & switch branch

git merge <branch> # Merge a branch into current branch
git pull origin <branch> # it first fetch(like check whether the branch has changed or not then Pull updates from that remote branch)
git fetch origin <branch> # Just tell if there are changes in that branch or not?
git push origin <branch> # Push local commits to remote

Stash: Stash is used to temporarily save all your changes when you don't want to push them to a branch but still want to keep your work safe.

Example: I'm working on the Admin branch, and my teammate is working on the Login branch. They have completed their work and raised a pull request. At that time, I don't want to push my work because it has a lot of errors.

If I don't stash my changes and try to pull the changes from the Login branch, the files from Login will merge with the files in Admin. If there are errors in the Login branch, it could affect my work and potentially cause me to lose my progress.

To prevent this, I first **stash** all my changes. Then I pull the changes from the Login branch and check if it works correctly. After reviewing, I can decide whether to keep the changes from the Login branch or not.

- If I want to keep my work, I **un-stash** my changes and continue working.

- If I don't want the changes from the Login branch, I can **revert** them and then **un-stash** my own work.

This ensures my progress is never lost while reviewing or merging others' code.

Commands:
```
git stash           # Temporarily save changes

git stash pop       # Apply and remove last stash
git stash apply     # Apply without removing stash
```

FETCH VS PULL:

- **Fetch:** Checks if the remote branch has changes and downloads them, but does **not merge** into your local branch.
- **Pull:** Checks for changes **and automatically merges** them into your current branch (fetch + merge).

[PAST PAPERS]

Q: Consider that you are working on a new feature for a popular shopping store to predict customer retention. The feature is almost half-way through and you are well on track to complete it by the end of the sprint. Meanwhile, your client requests an urgent fix for a security issue that is preventing the customers from signing into the application. Making good use of Git Version Control System, answer the following questions precisely:

a) What steps will you take to accommodate the client's request (for the security issue)?
Ans: I will stash(push to stack all changes done in that customer retention branch) so, all the changes should save. Then I will checkout to main/master branch and will create a new branch for security fix. Then do all work related to security fix. After all work I will raise a pull request and merge that branch into main/master branch. Then I will delete that security fix branch. Then I will pull all changes from main branch and test the security fix feature. If all is safe then I will un-stash my changes from the stack.

b) Steps after completing the customer retention feature at the end of sprint
Ans: I will raise a pull request and merge customer retention's branch into main/master branch and will delete this branch.

c) Git's role in billing the client
Ans :

- **Git logs and history** can show exactly what changes were made, by whom, and when. Like how many lines of code was added or deleted by whom.
- Commands like git log --author="Your Name" or git log --since="start_date" --until="end_date" can be used to summarize work for billing purposes.
- Feature branches and commits provide a **transparent record of effort**, useful for invoicing or client reports.

Q: A developer committed some unnecessary changes to an existing code file (Main.java) in the Git repository by mistake. (S)he wants to undo those changes and move the repository back to its previous state. What steps are required to accomplish this goal? Describe the strategy in general.
Ans: git reset --hard "commit-number(SHA) where you want to move"
Eg: git reset --hard 30cb4a3d43b1e31734097eedbe72ea5d78ab3bb4
Or we can do git rest –hard HEAD~N (where N is how many commits do you want to remove)
Eg: git reset --hard HEAD~1 (1 commit will remove)
It will remove all changes and commits from the local repo.

then , we have to do, git push origin HEAD --force.
Now the remote repo will also be updated.

Q: What are the advantages of using Git?
Ans:

- Distributed version control – multiple clones and offline commits.
- Branching and merging – easy to isolate features/fixes.
- History tracking – full audit trail of changes.
- Collaboration – multiple developers can work concurrently.
- Backup – every clone is a full repository.
- Revert & recovery – mistakes can be undone safely.

Q: what problems can occur while merging one branch to another and how to handle them?
Ans: Imagine the `main` branch has a file `Main.java`, and two developers are working on separate branches.

- The first developer completes their work and merges their branch into `main`.
- The second developer, who hasn't pulled the latest changes from `main`, tries to merge their branch.
- A warning or **merge conflict** occurs because the second developer's branch is behind `main`.
- If `Main.java` was modified by the first developer, Git cannot automatically merge, as the second developer has an outdated version.

Solution:

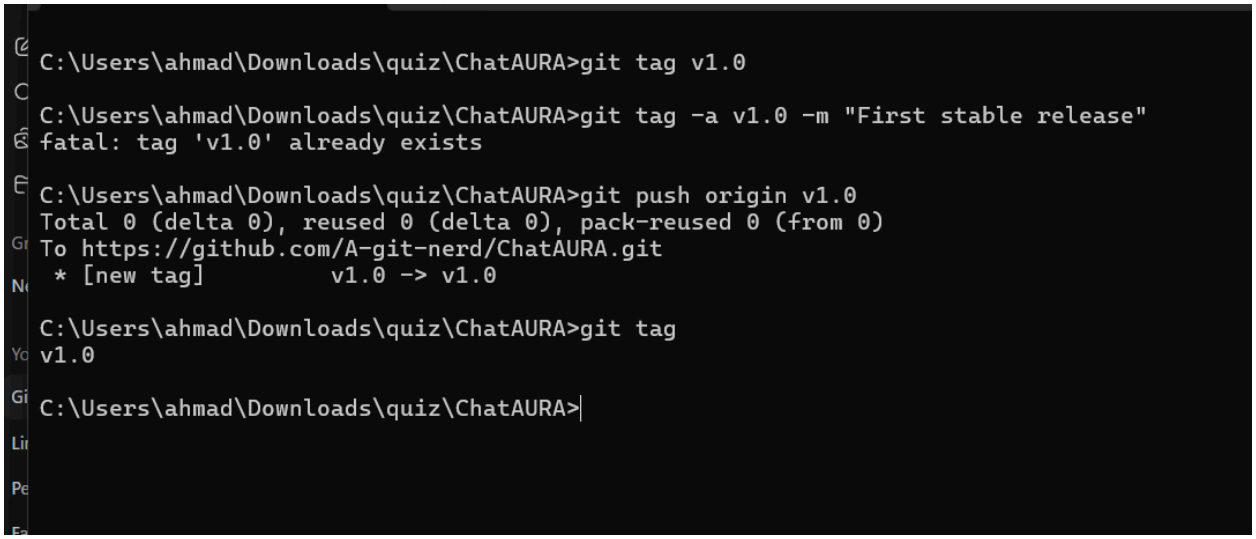
1. The second developer should first pull the latest changes from `main` into their branch.
2. If conflicts arise, manually resolve them by editing the files to choose which changes to keep.
3. Stage and commit the resolved files.
4. After resolving conflicts, the branch can be safely merged into `main`.

[CONCEPT]

Every commit has a unique SHA, like `a1b2c3d4e5`. And SHA used to revert changes to particular commit (See that qs where we delete the commit)

Tags are also kinda commits. Think about Heading and sub headings where headings are tags and sub headings are commits with SHA id.
We make different versions tags of the project so, if one version becomes unstable we can compare or revert the project to previous or some particular tag.

eg:



Q: What is the purpose of using tags in Git version control system? Explain in comparison with commit messages and revision numbers.

Ans:

Purpose of tags:

- Tags are used to mark specific points in the Git history as important, such as a release version or milestone.
- They provide a human-readable reference for commits, making it easier to identify significant states of the project.

Comparison:

- **Commit messages:** Describe what changes were made in a specific commit. They help understand the purpose of individual commits but do not mark a milestone.
- **Revision numbers (SHA):** Unique identifiers for each commit. They are precise but not easy to remember or reference.
- **Tags:** Human-friendly labels that point to a specific commit, such as “v1.0” or “release-2025”. They are easier to reference than SHAs and indicate important versions, unlike commit messages which describe changes but do not indicate milestones.

Comment:

- Commit messages explain **what** changed.
- Revision numbers provide a technical reference for each commit.
- Tags provide a clear, memorable label for versions or releases, making them ideal for tracking milestones, sharing with clients, or rolling back to specific versions