Open Ended Questions

- Explain which type of Git object is used to store the contents of a file and how this object fits into Git's object model.
  - Blob object
  - A blob contains blob word + file size + file content as hashed (no filename)
  - Blobs are grouped in **tree objects**

- Git allows configuration at system, global, and local levels. Explain which level takes priority if the same setting is defined in multiple places, and why this design is useful.
  - Local > Global > System
  - Useful because you can override project settings without affecting other repositories

---

- Compare `.gitignore` and `.git/info/exclude`. How are they similar, and in what situations would you use one instead of the other?
  - `.gitignore` → shared with the repository
  - `.git/info/exclude` → not shared
  - Using `.git/info/exclude` for my private data (like TODO list)

- What is the difference between `git diff` and git `diff --staged`? Describe a scenario where each command would be useful.
  - `git diff` → Difference between working tree and index (stage area)
  - `git diff --staged` → Difference between index (stage area) and Repo
  - Scenarios
    - Before staging → use `git diff`
    - After staging → use `git diff --staged`

- If you accidentally staged a file, how would you remove it from the staging area but keep your modifications in the working directory? Explain why this might be necessary.
  - `git restore --staged <fileName>` → file at index = file at Repo
  - If staged file, and want to edit before commit

- Can you directly alias git commit as git ci using Git configuration? Why or why not? If not, what alternatives exist?
  - No, because Git doesn't allow arbitrary shorthand for its **built-in subcommands**
  - alternatives → `git config --global alias.ci commit`

- What does the `init.defaultBranch` setting control in Git, and why might teams choose to set it differently?
  - Sets the default branch name when running `git init`
  - For different workflows and conventions

- Every commit in Git points to at least one tree object. Explain what this means and why it is important for Git's structure.
  - A tree represents the project snapshot (directories + blobs)
  - Commits reference trees to restore the project state at that point
  - This makes Git efficient at reconstructing history

- If you have staged changes in main and then switch to a feature branch, what happens to those staged changes? Why does Git behave this way?
  - Staged changes **carry over** to the new branch.
  - Reason
    - staging area is repository-wide, not tied to a branch.

- Both `git switch -c feature` and `git checkout -b feature` create a new branch. Explain the difference between these two commands and why Git introduced switch.
    - `checkout` does many things (branch switching, file restore, etc.) and can be confusing
    - `switch` was introduced to be clearer, dedicated only to branch operations

---