Task3: Version Control With Git/Github

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.
- 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.
- Compare `.gitignore` and `.git/info/exclude`. How are they similar, and in what situations would you use one instead of the other?
- What is the difference between `git diff` and `git diff --staged`? Describe a scenario where each command would be useful.
- 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.
- Can you directly alias `git commit` as `git ci` using Git configuration? Why or why not? If not, what alternatives exist?
- What does the `init.defaultBranch` setting control in Git, and why might teams choose to set it differently?
- Every commit in Git points to at least one tree object. Explain what this means and why it is important for Git's structure.
- 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?
- 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`.

MCQ Questions:

1. Which of the following is **NOT** a benefit of using version control?
   a) Collaboration among multiple developers
   b) Tracking changes and history
   c) Automatic bug fixing
   d) Backup of source code

2. In a **Centralized Version Control System (CVCS)**, where is the version database stored?
   a) On every developer's local computer
   b) On a single central server
   c) Only in the cloud
   d) In the `.git` folder

3. Who developed Git and in which year?
   a) Richard Stallman, 1991
   b) Linus Torvalds, 2005
   c) Dennis Ritchie, 1989
   d) Guido van Rossum, 1995

4. Which command checks the installed version of Git?
   a) `git config --version`
   b) `git --help`
   c) `git --version`
   d) `git show`

5. Which term refers to copying an existing remote repository to your local machine?
   a) Commit
   b) Fork
   c) Clone
   d) Pull

6. Which area in Git acts as an intermediate space between the working directory and the repository?
   a) Remote
   b) Staging area (index)
   c) Branch
   d) .gitignore

7. Which command renames the current branch to `main`?
   a) `git rename main`
   b) `git branch -M main`
   c) `git switch main`
   d) `git update main`

8. Which command is used to download a remote repository for the first time?
   a) `git clone`
   b) `git pull`
   c) `git push`
   d) `git fetch`

9. What is the purpose of a pull request in GitHub?
   a) To copy a repository from one account to another
   b) To suggest merging changes from one branch into another
   c) To delete a branch
   d) To reset the commit history

10. If Peter wants to push changes to Daniel's repository but doesn't have permission, what must Daniel do?
    a) Share his GitHub password with Peter
    b) Add Peter as a collaborator
    c) Run `git push --force`
    d) Delete and recreate the repository

Practice Project

Scenario

You are creating a small Python project with these files:

- `main.py`
- `utils/math_utils.py`
- `README.md`

Tasks

1. Setup

- Initialize a new Git repo.
- Configure your **default editor** (pick `nano`, `vim`, or `code --wait`).
- Add an **alias** so you can type `st` instead of the full status command.

---

2. First Commit

- Add all project files and make your **first commit**: *"Initial project structure"*.
- Explore the `.git/objects/` directory. (Hint: use a Git plumbing command to read the content of a blob or tree (cat-file)).

---

3. Ignore Files

- Create a rule to ignore all `log` files.
- Test by creating a `debug.log` file and check that Git ignores it.

---

4. New Feature (Branching)

- Create a new branch called `feature-math`.
- Inside `utils/math_utils.py`, add a function:

  ```
  def add(a, b):
  ```

```
    return a + b
```

- Commit this change to the branch.

---

5. Merging

- Switch back to `main`.
- Merge the branch into `main`.
- Check if the merge was fast-forward or a 3-way merge and what is the difference between the two ways and show your answer using a diagram or using (log command <-- bonus) ?

---

6. Undo / Unstage

- Edit `README.md` (e.g., add "This is a math project") and stage it.
- Oops! Unstage it without deleting your changes.
- Then discard your changes completely.
- Explain for me what is the difference between restore –staged, --worktree and rm –cached ? And show your explanation in your terminal <3.

---

7. Bonus Challenge

- Ignore a file using `.git/info/exclude` instead of `.gitignore.`
- Visualize the commit history as a **graph** (Hint: compact one-line graph view).