Name : Ahmed Khaled Abdelmaksod

Group : IOT_701_O

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 uses a Blob object to store the contents of a file. Blobs hold the raw file data without filenames or metadata. Trees map filenames to blobs (and subtrees), commits point to trees (representing a snapshot of the project), and tags provide references to commits.

- **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.**

– Git supports three main levels of configuration:

1. **System level**
   - File: /etc/gitconfig → applies to all users on the system.
2. **Global level**
   - File: ~/.gitconfig → applies only to the current user
3. **Local level**
   - File: .git/config inside the repository → applies only to that repository.

– priority from highest to low => Local → Global → System

– This layered design is useful because it allows system-wide defaults, user-specific customization, and project-specific overrides, giving developers maximum flexibility without losing consistency.

- **Compare .gitignore and .git/info/exclude. How are they similar, and in what situations would you use one instead of the other?**

Both specify files Git should ignore. The difference is scope: .gitignore is versioned and shared across the team (for project-wide ignores), while .git/info/exclude is local for personal ignores.

- **What is the difference between git diff and git diff --staged? Describe a scenario where each command would be useful.**

– git diff shows unstaged changes (working directory vs staging area).

– git diff --staged shows staged changes (staging area vs last commit).
This distinction is useful for reviewing what's not yet staged versus what's staged and about to be committed.

- **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.**

– We can run 'git reset HEAD <file>' to unstage a file while keeping its modifications. This is necessary when we accidentally stage files we don't want in the next commit, allowing we to control commit contents without losing our work.

- **Can you directly alias git commit as git ci using Git configuration? Why or why not? If not, what alternatives exist?**

we cannot completely replace Git's built-in commit command, but we can alias it inside Git with: git config --global alias.ci commit

to use git ci. If we want ci as a standalone command, we can use a shell alias or a wrapper script.

- **What does the init.defaultBranch setting control in Git, and why might teams choose to set it differently?**

The init.defaultBranch setting controls the name of the initial branch created by git init. Teams may choose different values (e.g., main, develop, trunk) for reasons such as inclusivity, workflow conventions, tool compatibility, or consistency with legacy projects.

- **Every commit in Git points to at least one tree object. Explain what this means and why it is important for Git's structure.**

Every commit in Git points to a tree object representing the root directory of the repository at that moment. This ensures that each commit corresponds to a complete project snapshot. It's crucial because it makes Git's model snapshot-based, ensures data integrity via content-addressing, allows efficient reuse of unchanged objects, and enables easy history comparison.

- **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 remain staged when you switch branches, and if you commit, they will be recorded in the new branch. Git behaves this way because the staging area (index) is shared

across branches, ensuring predictability and giving you flexibility to move staged work between branches.

- **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.**

Both git checkout -b feature and git switch -c feature create and switch to a new branch. The difference is that checkout is an older, overloaded command that handles multiple tasks (branch switching, file restoration, branch creation), while switch was introduced to make Git clearer and safer by handling only branch-related actions.

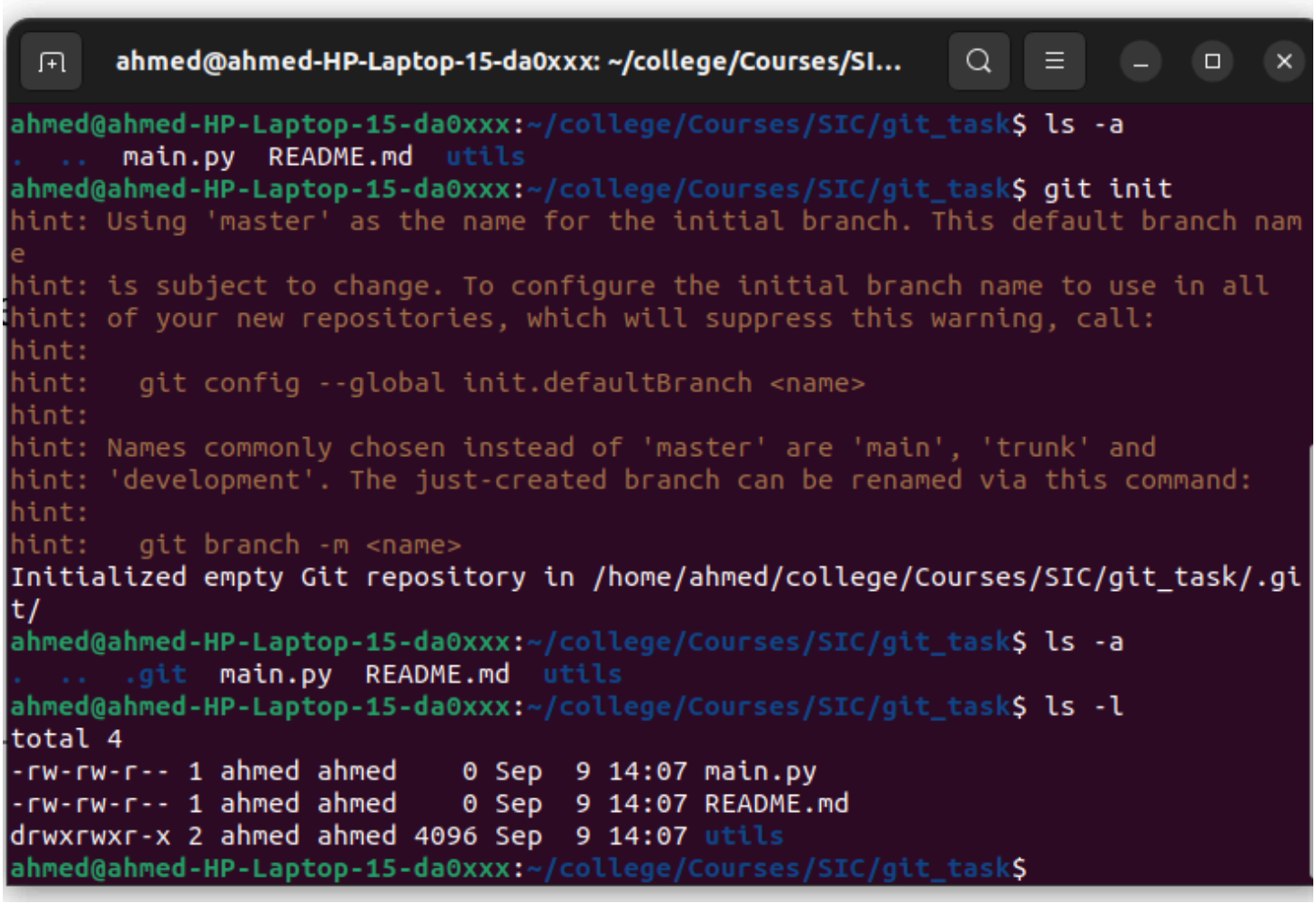
MCQ Questions:

1. C
2. B
3. B
4. C
5. C
6. B
7. B
8. A
9. B
10. B

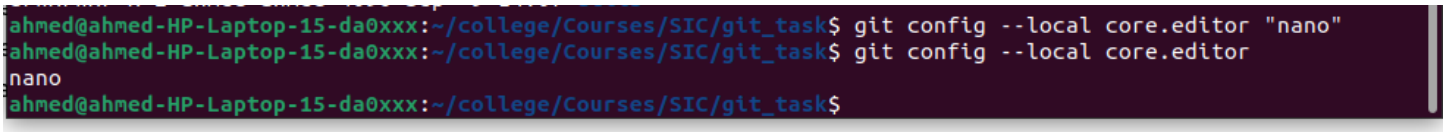
Practice Project 🙂

1. Setup

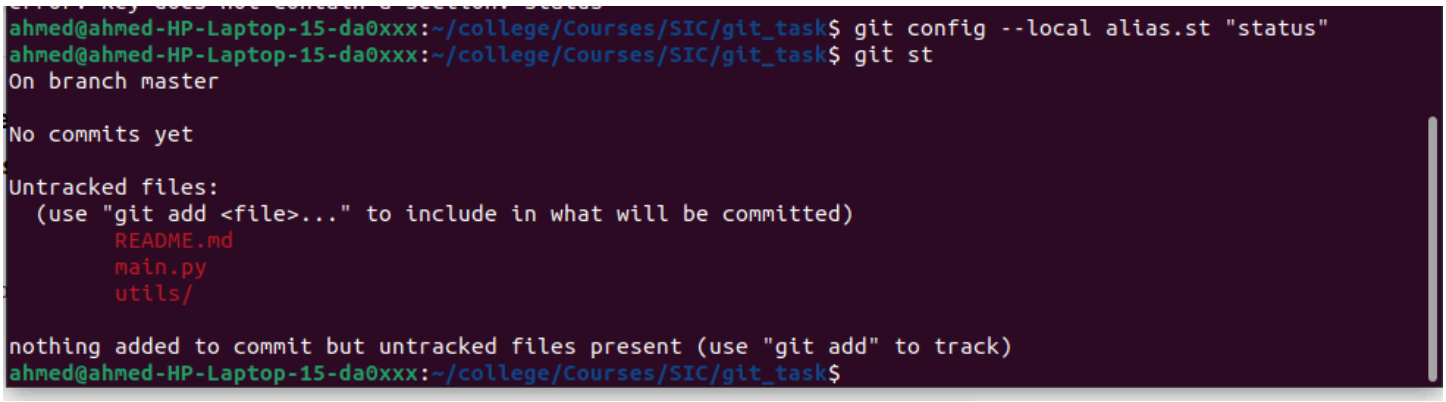
• Initialize a new Git repo.



- **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”.**

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git add .
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git commit -m "Initial project structure"
[master (root-commit) d39b970] Initial project structure
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
 create mode 100644 main.py
 create mode 100644 utils/math_utils.py
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$
```

• Explore the .git/objects/ directory. (Hint: use a Git plumbing command to read the content of a blob or tree (cat-file)).

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cd .git/objects/
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git/objects$ git log --oneline
d39b970 (HEAD -> master) Initial project structure
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git/objects$ git cat-file -t d39b970
commit
```

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git/objects$ git cat-file -p d39b970
tree 34cb52607e8879100035ed018a27cc89e581bcaa
author ahmed-khaled-abdelmaksod <ak4902280@gmail.com> 1757416917 +0300
committer ahmed-khaled-abdelmaksod <ak4902280@gmail.com> 1757416917 +0300

Initial project structure
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git/objects$ ls -l
total 24
drwxrwxr-x 2 ahmed ahmed 4096 Sep  9 14:21 34
drwxrwxr-x 2 ahmed ahmed 4096 Sep  9 14:21 8a
drwxrwxr-x 2 ahmed ahmed 4096 Sep  9 14:21 d3
drwxrwxr-x 2 ahmed ahmed 4096 Sep  9 14:21 e6
drwxrwxr-x 2 ahmed ahmed 4096 Sep  9 14:08 info
drwxrwxr-x 2 ahmed ahmed 4096 Sep  9 14:08 pack
```

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git/objects/34$ git cat-file -p 34cb52607e8
879100035ed018a27cc89e581bcaa
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    README.md
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    main.py
040000 tree 8af8210a0ad9f3e2a137eee5dffcf551c141fcb1    utils
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git/objects/34$ git cat-file -p 8af8210a0ad
9f3e2a137eee5dffcf551c141fcb1
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    math_utils.py
```

• Create a rule to ignore all log files.

• Test by creating a debug.log file and check that Git ignores it.

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git/objects/34$ cd ../../
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git$ cd ..
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ touch .gitignore
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ nano .
./          ../         .git/       .gitignore
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ nano .gitignore
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cat .gitignore
*.log
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ nano debug.log
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cat debug.log
Ahmed Khaled
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git st
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git add .
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git commit -m "Add .gitignore file"
[master 8bb1cfb] Add .gitignore file
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git st
On branch master
nothing to commit, working tree clean
```

```
nothing to commit, working tree clean
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git log --all
commit 8bb1cfbaac8b3619e82a93468c651045773260d6 (HEAD -> master)
Author: ahmed-khaled-abdelmaksod <ak4902280@gmail.com>
Date:   Tue Sep 9 14:51:44 2025 +0300

    Add .gitignore file

commit d39b970d4d8114e14656f179a74536d2cd927b83
Author: ahmed-khaled-abdelmaksod <ak4902280@gmail.com>
Date:   Tue Sep 9 14:21:57 2025 +0300

    Initial project structure
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git cat-file -p 8bb1cfbaac8b3619e82a93468c6
51045773260d6
tree f34ae8560fedbdac63ec1fd00947c9aa58518ee3
parent d39b970d4d8114e14656f179a74536d2cd927b83
author ahmed-khaled-abdelmaksod <ak4902280@gmail.com> 1757418704 +0300
committer ahmed-khaled-abdelmaksod <ak4902280@gmail.com> 1757418704 +0300

Add .gitignore file
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$

ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git cat-file -p f34ae8560fedbdac63ec1fd0094
7c9aa58518ee3
100644 blob 397b4a7624e35fa60563a9c03b1213d93f7b6546    .gitignore
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    README.md
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    main.py
040000 tree 8af8210a0ad9f3e2a137eee5dffcf551c141fcb1    utils
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ ls -l
total 8
-rw-rw-r-- 1 ahmed ahmed   13 Sep  9 14:51 debug.log
-rw-rw-r-- 1 ahmed ahmed    0 Sep  9 14:07 main.py
-rw-rw-r-- 1 ahmed ahmed    0 Sep  9 14:07 README.md
drwxrwxr-x 2 ahmed ahmed 4096 Sep  9 14:07 utils
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$
```

---

4. New Feature (Branching)

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

Commit this change to the branch.

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git switch -c feature-math
Switched to a new branch 'feature-math'
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git branch
* feature-math
  master
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ nano utils/math_utils.py
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cat utils/math_utils.py
def sum(a,b):
    return a + b;
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git add .
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git commit -m "Add sum function"
[feature-math d5f02d0] Add sum function
 1 file changed, 2 insertions(+)
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$
```

---

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) ?

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git switch master
Switched to branch 'master'
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git branch
  feature-math
* master
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cat utils/math_utils.py
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git merge feature-math
Updating 8bb1cfb..d5f02d0
Fast-forward
 utils/math_utils.py | 2 ++
 1 file changed, 2 insertions(+)
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cat utils/math_utils.py
def sum(a,b):
    return a + b;
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$
```

Fast forward because there are no conflicts.

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git log --all --oneline --graph
* d5f02d0 (HEAD -> master, feature-math) Add sum function
* 8bb1cfb Add .gitignore file
* d39b970 Initial project structure
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$
```

---

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.

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ nano README.md
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git add .
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git st
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md

ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git restore --staged README.md
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git st
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cat README.md
This is a math project
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git restore README.md
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cat README.md
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ Discarded :))
```

restore --staged → unstages but keeps edits.

restore --worktree → discards edits, restores file content.

rm --cached → stops tracking file entirely, keeps it on disk.

---

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).

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ touch try.c
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ ls
debug.log  main.py  README.md  try.c  utils
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ ls .git/info
exclude
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cat .git/info/
cat: .git/info/: Is a directory
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cat .git/info/
cat: .git/info/: Is a directory
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cd .git/info/
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git/info$ ls -l
total 4
-rw-rw-r-- 1 ahmed ahmed 240 Sep  9 14:08 exclude
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git/info$ nano exclude
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task/.git/info$ cd ../../../
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC$ cd git_task/
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git st
On branch master
nothing to commit, working tree clean
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ cat .git/info/exclude
# git ls-files --others --exclude-from=.git/info/exclude
# Lines that start with '#' are comments.
# For a project mostly in C, the following would be a good set of
# exclude patterns (uncomment them if you want to use them):
# *.[oa]
# *~
*.c
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$
```

```
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$ git log --oneline --graph --decorate --all
* d5f02d0 (HEAD -> master, feature-math) Add sum function
* 8bb1cfb Add .gitignore file
* d39b970 Initial project structure
ahmed@ahmed-HP-Laptop-15-da0xxx:~/college/Courses/SIC/git_task$
```