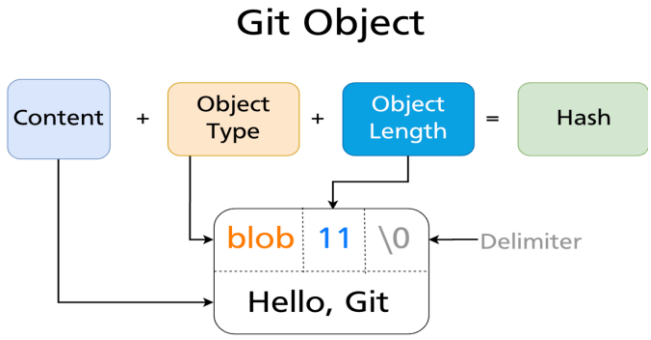
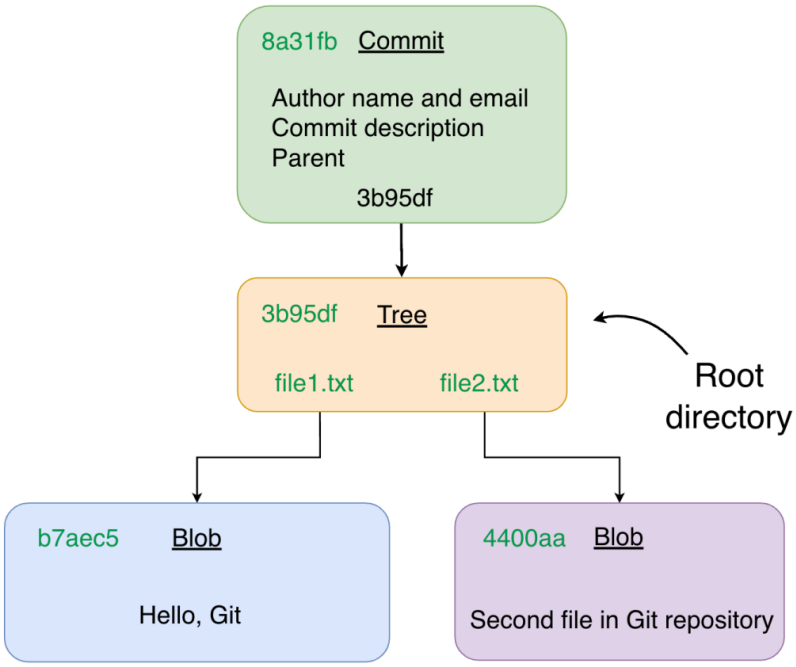
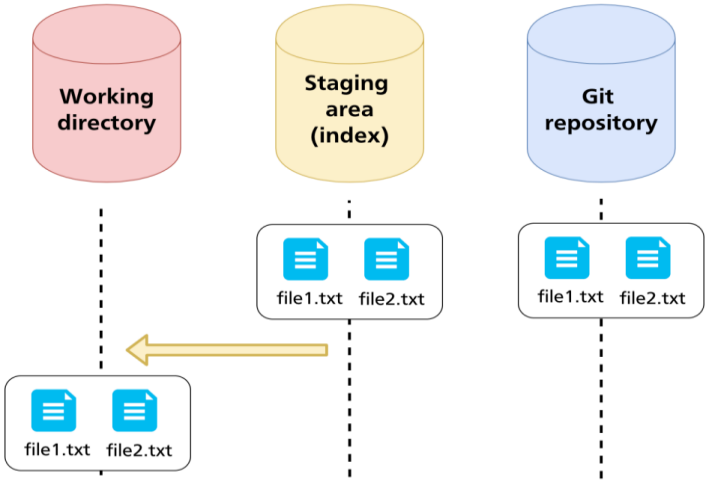
DEVOPS - LAB SET QUESTIONSs

<table>
<tr><th>Q No</th><th>Lab Set - Question Bank</th></tr>
<tr><td>1</td><td><u>SDLC Life Cycle</u>: Learn and understand Agile Life Cycle Process and Practices
<u>Linux Exercise:</u> Exercise File Management
I. a) Create 10 files under your home directory (File names = cse, ise, aiml, ai_ds, ece, eee, eie, iem, civil, and mech)
b) Create 3 directories under your home directory (Dir name = itbranch, circuitbranch and others)
c) Create a new file students and write to it as "Students of SIT". Then create a soft link in /tmp directory. Also create a hard link of students in /tmp directory. Check the inodes of both links
II. a) Create a script to the output of ls -ltr to a file called golisting
b) Create a script to output "First day of Odd Sem 3rd Sem is dd/mm/yyyy" (note: display dd/mm/yyyy with values)
c) Write a shell script that will look at your current day and tell you the state of the backup</td></tr>
<tr><td>2</td><td>I. Create following git object and Analyze Tree Objects and folder structure.
 

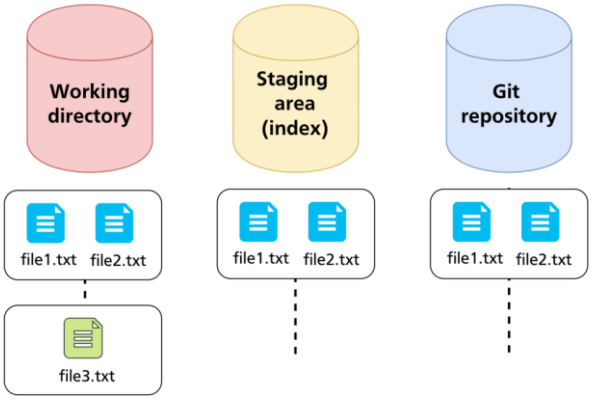
II. Implement the following
 

(Note: the SHA1 hash number is mentioned only for reference here as it may vary)</td></tr>
<tr><td>3</td><td>I. Exercise Git files Life Cycle and track the complete status (Untracked, Modified, Staged, Unmodified)</td></tr>
</table>

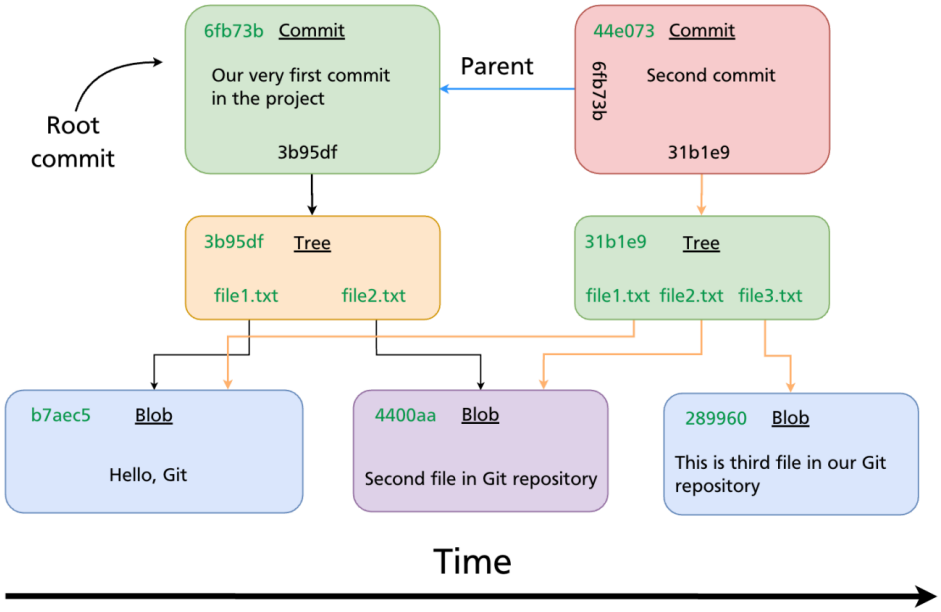
II. Use Git commands and build the following scenario



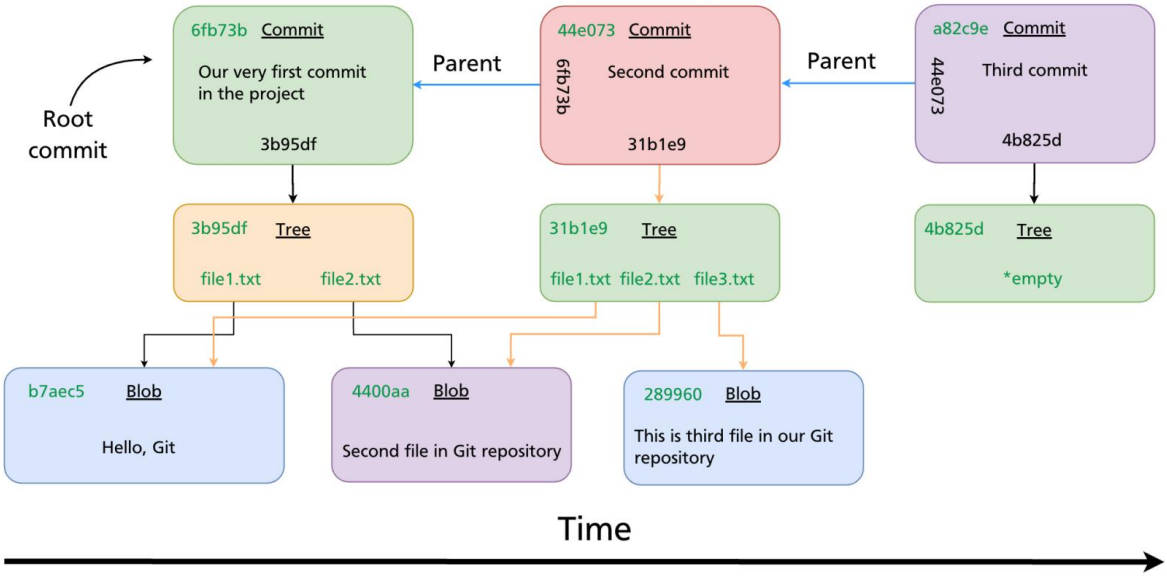
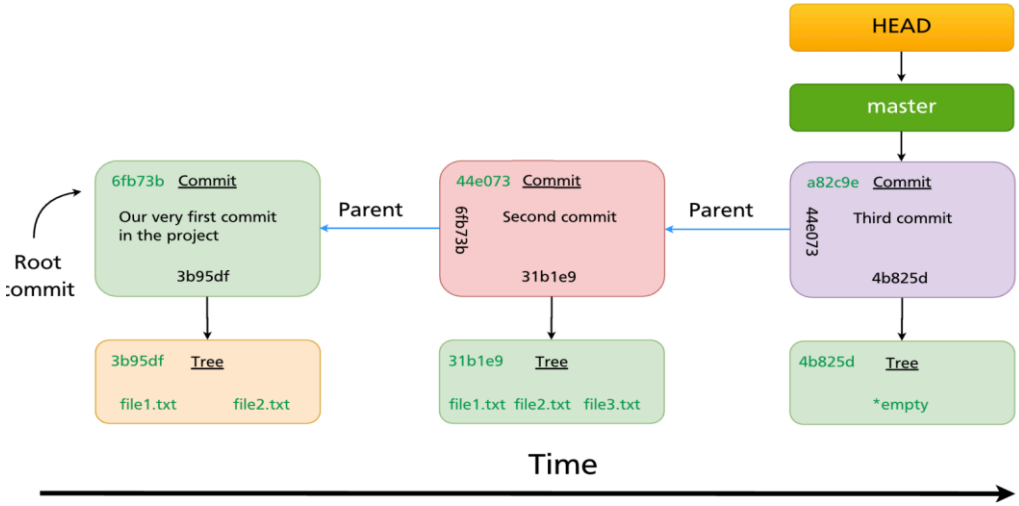
III. Perform the git operations to build following scenario i.e, create new file file3.txt in local working directory and then add it to Git repository. Track the file status.

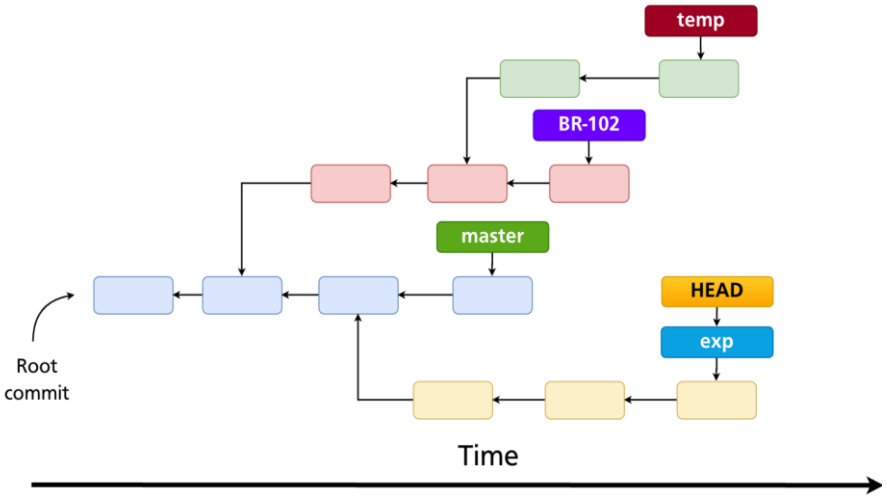


4. I. Implement the following:



(Note: the SHA1 hash number is mentioned only for reference here as it may vary)

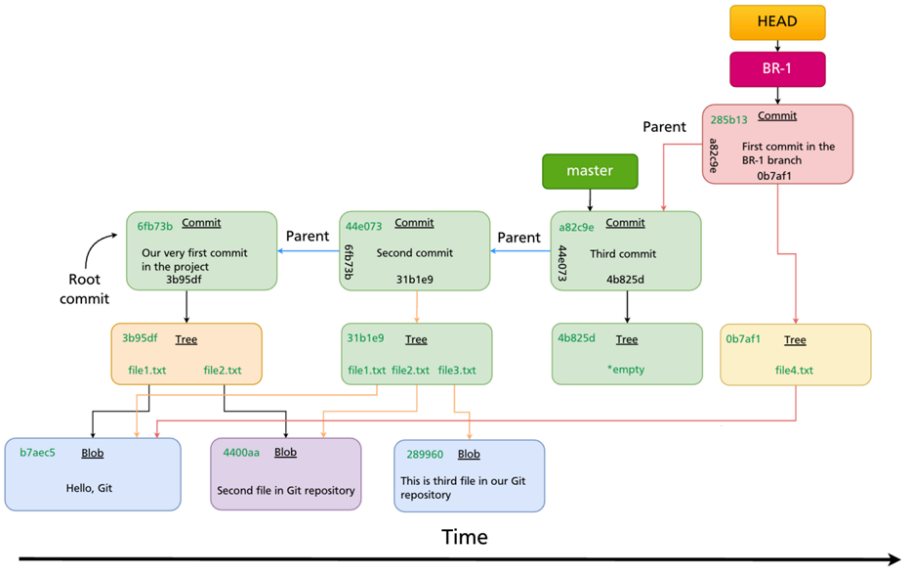
	II. Implement the following:  (Note: the SHA1 hash number is mentioned only for reference here as it may vary) Compare the commits and analyze the differences.
5	Implement the following: (Note: the SHA1 hash number is mentioned only for reference) I.  II.



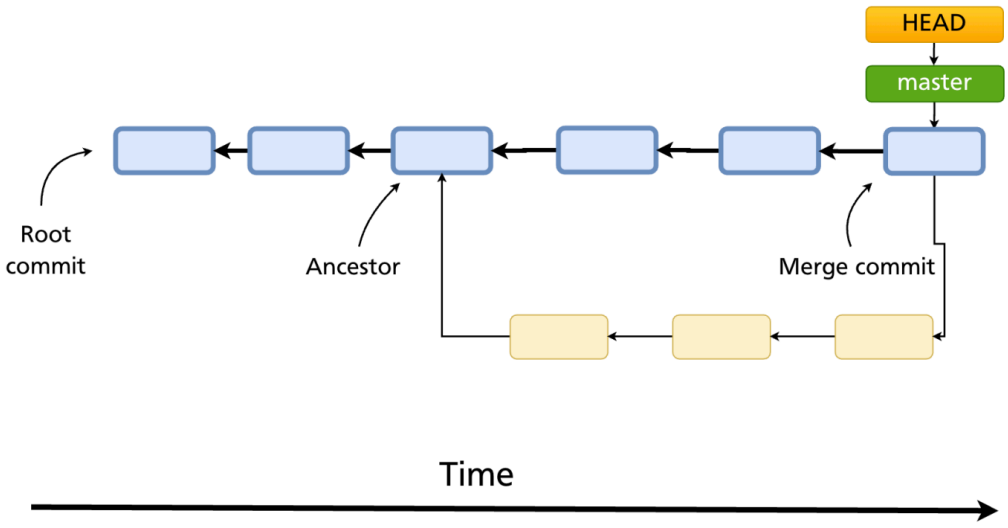
Additional: Introduce merge conflicts and resolve

6

I. Implement the following: (Note: the SHA1 hash number is mentioned only for reference)



II. Perform 3 way merge as per following scenario. Introduce conflicts and resolve merge conflict.



	Additonal: Do a project management activity by creating a simple project. Add 2 to 3 collaborators and Perform Git collaboration activity. **Scenario:** Two developers modify the same line in App.java on separate branches. You need to merge their changes and manually resolve conflicts. Clone, Modify, and Collaborate on the Project.
7	Set up and execute a Maven-based Java project build using Jenkins. Maven project should successfully build and generates a .jar file inside the Jenkins workspace. Configure Jenkins to automatically build when code is pushed to GitHub. Ensure each Git commit triggers a Jenkins build automatically.
8	An organization uses Jenkins to build a Java and Maven project stored on GitHub. A project manager requests that builds should happen automatically when developers push code and also once every night at 2 AM. Write the appropriate cron syntax for the nightly build. Create appropriate Jenkins job to handle the scenario.
9	You are given a simple **Java application** named Hello.java that prints the message **"Hello from Docker Container!"**. Your task is to **create a Docker image**, **run the container**, and **analyze** its behavior if any issues occur during the build or execution process. I. Write a **Dockerfile** to containerize your Hello.java application. II. Use **Docker commands** to build an image and run it as a container. III. If the container does not display the expected message, **analyze the reason**, fix the issue, and demonstrate a successful output
10	You have created a **Java Login Application** (LoginApp.java) that accepts a **username and password** as input and prints "Login Successful" if credentials match predefined values, otherwise prints "Access Denied." Your task is to **containerize** this Java program using **Docker**, **build and run** the container, and **analyze** issues if the program fails during build or runtime I. Write a **Dockerfile** to build a Docker image for the LoginApp.java program. II. Execute **Docker commands** to build the image, run the container, and display program output. III. If any errors occur (for example, javac not finding the file or incorrect input handling), **analyze the cause**, fix it, and re-run the container successfully.