PES Institute of Technology and Management
NH-206, Sagar Road, Shivamogga-577204

Department of Information Science and Engineering

Affiliated to
VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Jnana Sangama, Belagavi, Karnataka –590018

Laboratory Manual

III Semester B.E Project Management with Git (BCS358C)

Prepared By ,

Mrs. H. Manjula,
Assistant Professor,
Department of CSE

Department Vision

To create technically competent, ethically strong and skilled IT Professionals.

Department Mission

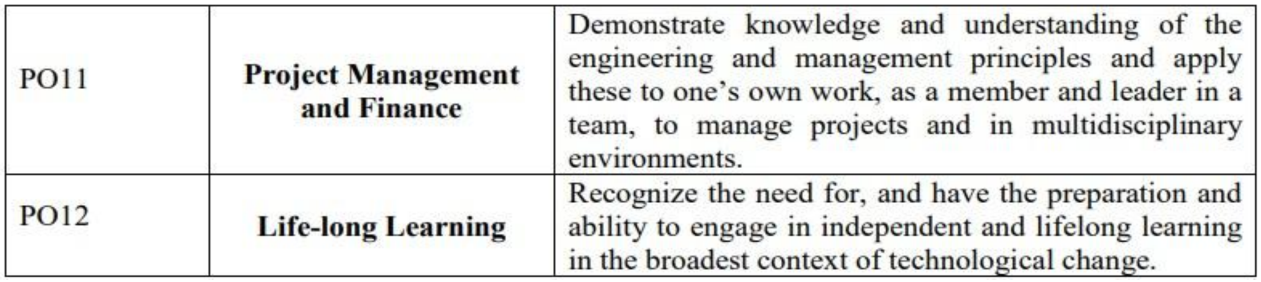
M1: Enhancing the knowledge through ongoing research and professional development programs by incorporating effective teaching and learning methods.

M2: To prepare students to be successful in their professional career and continuous learners by offering a strong technical foundation with professional and Ethical responsibilities.

Program Outcomes as defined by NBA (PO)

Engineering Graduate will be able to:

Programme Outcome	PO Type	Engineering graduates will be able to
PO1	**Engineering Knowledge**	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem Analysis**	Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
PO3	**Design/Development of Solutions**	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct Investigations of Complex Problems**	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions for complex problems.
PO5	**Modern Tool Usage**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
PO6	**The Engineer and Society**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and Sustainability**	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and Team Work**	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.



PO11	**Project Management and Finance**	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long Learning**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

CONTENTS

Project Management with Git		Semester	3
Course Code	BCS358C	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0: 0 : 2: 0	SEE Marks	50
Credits	01	Exam Marks	100
Examination type (SEE)	Practical		

Course objectives:

- .To familiar with basic command of Git
- To create and manage branches
- To understand how to collaborate and work with Remote Repositories
- To familiar with virion controlling commands

Sl.NO	Experiments
1	**Setting Up and Basic Commands** Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.
2	**Creating and Managing Branches** Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."
3	**Creating and Managing Branches** Write the commands to stash your changes, switch branches, and then apply the stashed changes.
4	**Collaboration and Remote Repositories** Clone a remote Git repository to your local machine.
5	**Collaboration and Remote Repositories** Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.
6	**Collaboration and Remote Repositories** Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.
7	**Git Tags and Releases** Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.
8	**Advanced Git Operations**

	Write the command to cherry-pick a range of commits from "source-branch" to the current branch.
9	**Analysing and Changing Git History** Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?
10	**Analysing and Changing Git History** Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."
11	**Analysing and Changing Git History** Write the command to display the last five commits in the repository's history.
12	**Analysing and Changing Git History** Write the command to undo the changes introduced by the commit with the ID "abc123".

Course outcomes (Course Skill Set):
At the end of the course the student will be able to:

- Use the basics commands related to git repository
- Create and manage the branches
- Apply commands related to Collaboration and Remote Repositories
- Use the commands related to Git Tags, Releases and advanced git operations
- Analyse and change the git history

Experiments on Project Management with Git

Experiment 1.

Setting Up and Basic Commands:

Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.

Solution:

To initialize a new Git repository in a directory, create a new file, add it to the staging area, and commit the changes with an appropriate commit message, follow these steps:

1. Open your terminal and navigate to the directory where you want to create the Git repository.
2. Initialize a new Git repository in that directory:

```
$ git init
```

3. Create a new file in the directory. For example, let's create a file named "my_file.txt "You can use any text editor or command-line tools to create the file.
   (touch my_file.txt).

4. Add the newly created file to the staging area. Replace "my_file.txt" with the actual name of your file:

```
$ git add my_file.txt
```

This command stages the file for the upcoming commit.

5. Commit the changes with an appropriate commit message. Replace "Your commit message here “with a meaningful description of your changes:

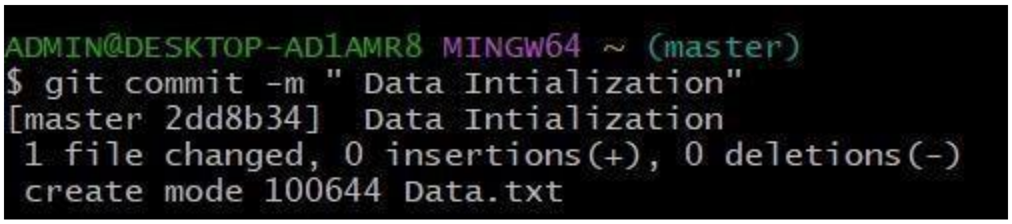
```
$ git commit -m "Your commit message here"
```

Your commit message should briefly describe the purpose or nature of the changes you made.
For example:

```
$ git commit -m "Add a new file called my_file.txt"
```

After these steps, your changes will be committed to the Git repository with the provided commit message. You now have a version of the repository with the new file and its history stored in Git.

Output:



```
ADMIN@DESKTOP-AD1AMR8 MINGW64 ~ (master)
$ git commit -m " Data Intialization"
[master 2dd8b34]  Data Intialization
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Data.txt
```

Experiment 2

Creating and Managing Branches:

Create a new branch named "feature-branch." Switch to the "master" branch. Merge the"feature-branch" into "master."

Solution:

To create a new branch named "feature-branch," switch to the "master" branch, and merge the"feature-branch" into "master" in Git, follow these steps:

1. Make sure you are in the "master" branch by switching to it:

```
$ git checkout master
```

2. Create a new branch named "feature-branch" and switch to it:

```
$ git checkout -b feature-branch
```

This command will create a new branch called "feature-branch" and switch to it.

3. Make your changes in the "feature-branch" by adding, modifying, or deleting files asneeded.
4. Stage and commit your changes in the "feature-branch":

```
$ git add .
$ git commit -m "Your commit message for feature-branch"
```

Replace "Your commit message for feature-branch" with a descriptive commit message for thechanges you made in the "feature-branch."

5. Switch back to the "master" branch:

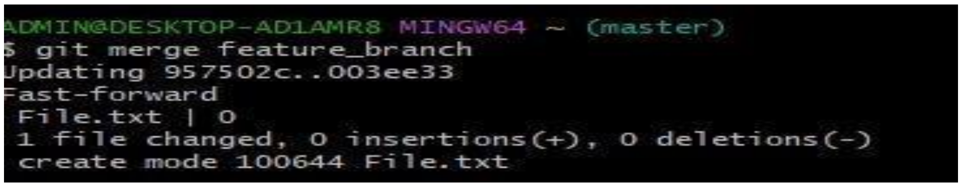
```
$ git checkout master
```

6. Merge the "feature-branch" into the "master" branch:

```
$ git merge feature-branch
```

This command will incorporate the changes from the "feature-branch" into the "master" branch.
Now, your changes from the "feature-branch" have been merged into the "master" branch. Yourproject's history will reflect the changes made in both branches.

Output:



```
ADMIN@DESKTOP-AD1AMR8 MINGW64 ~ (master)
$ git merge feature_branch
Updating 957502c..003ee33
ast-forward
 File.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 File.txt
```

Experiment 3

Creating and Managing Branches:

Write the commands to stash your changes, switch branches, and then apply the stashed changes.

Solution:

To stash your changes, switch branches, and then apply the stashed changes in Git, you can usethe following commands:

1. Stash your changes:

```
$ git stash save "Your stash message"
```

This command will save your changes in a stash, which acts like a temporary storage for changes that are not ready to be committed.

2. Switch to the desired branch:

```
$ git checkout target-branch
```

Replace "target-branch" with the name of the branch you want to switch to.

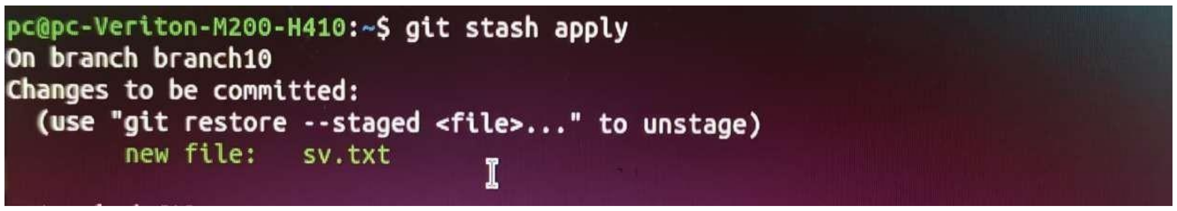
3. Apply the stashed changes:

```
$ git stash apply
```

This command will apply the most recent stash to your current working branch. If you have multiple stashes, you can specify a stash by name or reference (e.g., git stash apply stash@{2})if needed.

If you want to remove the stash after applying it, you can use git stash pop instead of git stash apply. Remember to replace "Your stash message" and "target-branch" with the actual message you want for your stash and the name of the branch you want to switch to

Output:



```
pc@pc-Veriton-M200-H410:~$ git stash apply
On branch branch10
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   sv.txt
```

Experiment 4

Collaboration and Remote Repositories:

Clone a remote Git repository to your local machine.

Solution:

To clone a remote Git repository to your local machine, follow these steps:

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to clone the remote Git repository. You can use the cd command to change your working directory.
3. Use the git clone command to clone the remote repository. Replace <repository_url> with the URL of the remote Git repository you want to clone. For example, if you werecloning a repository from GitHub, the URL might look like this:

```
$ git clone <repository_url>
```

Here's a full example:

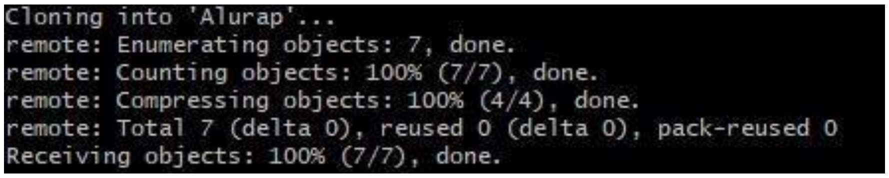
```
$ git clone https://github.com/username/repo-name.git
```

Replace https://github.com/username/repo-name.git with the actual URL of the repository youwant to clone.

4. Git will clone the repository to your local machine. Once the process is complete, youwill have a local copy of the remote repository in your chosen directory.

You can now work with the cloned repository on your local machine, make changes, and pushthose changes back to the remote repository as needed.

Output:



```
Cloning into 'Alurap'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), done.
```

Experiment 5

Collaboration and Remote Repositories: Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

Solution:

To fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch in Git, follow these steps:

1. Open your terminal or command prompt.’

2. Make sure you are in the local branch that you want to rebase. You can switch to the branch using the following command, replacing <branch-name> with your actual branch name:

```
$ git checkout <branch-name>
```

3. Fetch the latest changes from the remote repository. This will update your local repository with the changes from the remote without merging them into your local branch:

```
$ git fetch origin
```

Here, origin is the default name for the remote repository. If you have multiple remotes, replaceorigin with the name of the specific remote you want to fetch from.

4. Once you have fetched the latest changes, rebase your local branch onto the updated remote branch:

```
$ git rebase origin/<branch-name>
```

Replace <branch-name> with the name of the remote branch you want to rebase onto. This command will reapply your local commits on top of the latest changes from the remote branch, effectively incorporating the remote changes into your branch history.

5. Resolve any conflicts that may arise during the rebase process. Git will stop and notifyyou if there are conflicts that need to be resolved. Use a text editor to edit the conflicting files, save the changes, and then continue the rebase with:

```
$ git rebase –continue
```

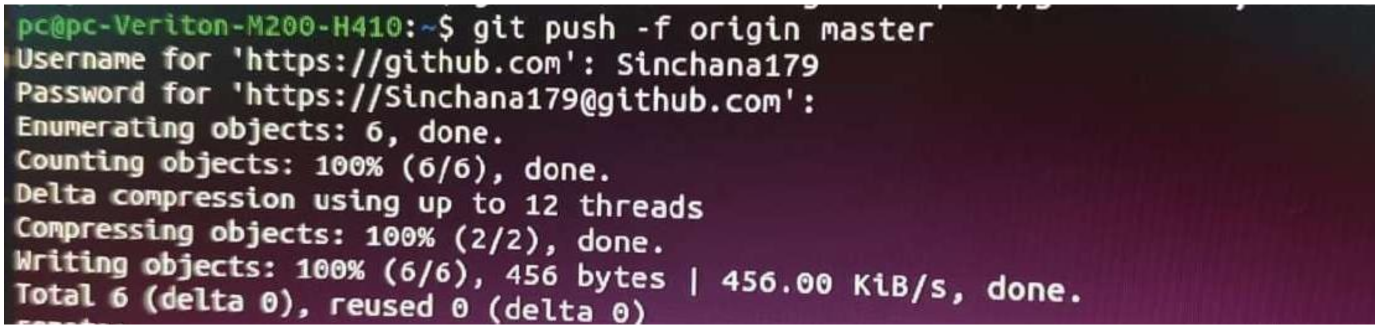
6. After resolving any conflicts and completing the rebase, you have successfully updatedyour local branch with the latest changes from the remote branch.

7. If you want to push your rebased changes to the remote repository, use the git push command. However, be cautious when pushing to a shared remote branch, as it can potentially overwrite otherdevelopers' changes:

```
$ git push origin <branch-name>
```

Replace <branch-name> with the name of your local branch. By following these steps, you cankeep your local branch up to date with the latest changes from the remote repository and maintain a clean and linear history through rebasing.

Output:



```
pc@pc-Veriton-M200-H410:~$ git push -f origin master
Username for 'https://github.com': Sinchana179
Password for 'https://Sinchana179@github.com':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 456 bytes | 456.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
```

Experiment 6

Collaboration and Remote Repositories:
Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

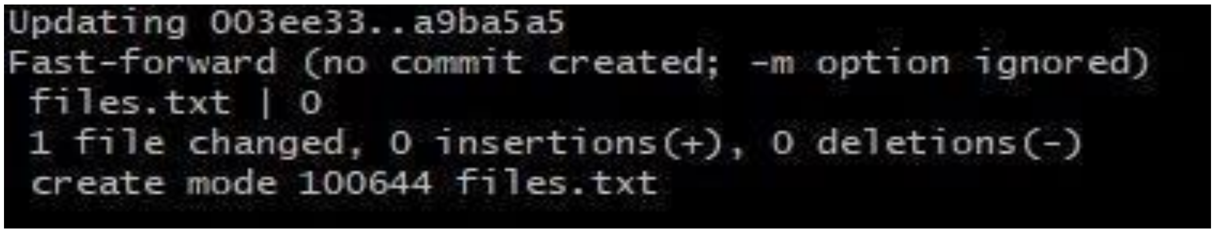
Solution:

To merge the "feature-branch" into "master" in Git while providing a custom commit messagefor the merge, you can use the following command:

```
$ git checkout master
$ git merge feature-branch -m "Your custom commit message here"
```

Replace "Your custom commit message here" with a meaningful and descriptive commit message for the merge. This message will be associated with the merge commit that is createdwhen you merge "feature-branch" into "master.

Output:



```
Updating 003ee33..a9ba5a5
Fast-forward (no commit created; -m option ignored)
 files.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 files.txt
```

Experiment 7.

Git Tags and Releases:
Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.

Solution:

To create a lightweight Git tag named "v1.0" for a commit in your local repository, you can use the following command:

```
$ git tag v1.0
```

This command will create a lightweight tag called "v1.0" for the most recent commit in your current branch. If you want to tag a specific commit other than the most recent one, you can specify the commit's SHA-1 hash after the tag name. For example:

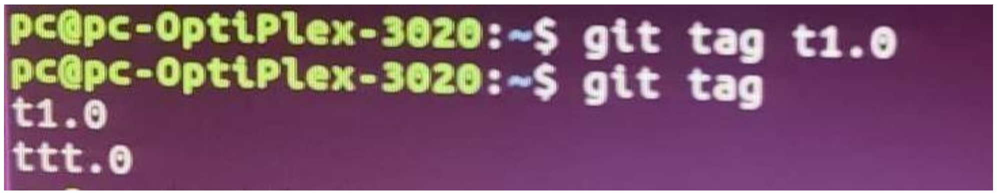
```
$ git tag v1.0 <commit-SHA>
```

Example:

```
$ git tag t1.0 <commit-SHA>
```

Replace <commit-SHA> with the actual SHA-1 hash of the commit you want to tag.

Output:



```
pc@pc-OptiPlex-3020:~$ git tag t1.0
pc@pc-OptiPlex-3020:~$ git tag
t1.0
ttt.0
```

Experiment 8.

Advanced Git Operations:
Write the command to cherry-pick a range of commits from "source-branch" to the current branch.

Solution:

To cherry-pick a range of commits from "source-branch" to the current branch, you can use the following command:

```
$ git cherry-pick <start-commit>^..<end-commit>
```

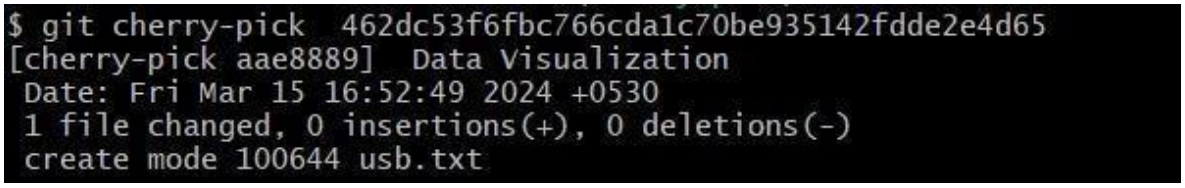
Replace <start-commit> with the commit at the beginning of the range, and <end-commit> with the commit at the end of the range. The ^ symbol is used to exclude the <start-commit> itself and include all commits after it up to and including <end-commit>. This will apply the changes from the specified range of commits to your current branch.

For example, if you want to cherry-pick a range of commits from "source-branch" starting fromcommit ABC123 and ending at commit DEF456, you would use:

```
$ git cherry-pick ABC123^..DEF456
```

Make sure you are on the branch where you want to apply these changes before running the cherry-pick command.

Output:



```
$ git cherry-pick  462dc53f6fbc766cda1c70be935142fdde2e4d65
[cherry-pick aae8889]  Data Visualization
 Date: Fri Mar 15 16:52:49 2024 +0530
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 usb.txt
```

Experiment 9.

Analysing and Changing Git History:
Given a commit ID, how would you use Git to view the details of that specific commit,including the author, date, and commit message?

Solution:

To view the details of a specific commit, including the author, date, and commit message, youcan use the git show or git log command with the commit ID. Here are both options:

1. Using git show:bash

```
$ git show <commit-ID>
```

Replace <commit-ID> with the actual commit ID you want to view. This command will display detailed information about the specified commit, including the commit message, author, date

example:

```
$ git show abc123
```

2. Using git log:

```
$ git log -n 1 <commit-ID>
```

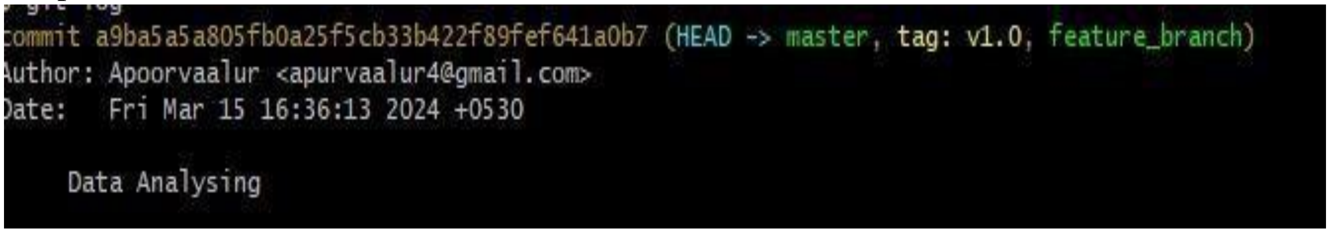
The -n 1 option tells Git to show only one commit. Replace <commit-ID> with the actual commit ID. This command will display a condensed view of the specified commit, including its commit message, author, date, and commit ID.

For example:

```
$ git log -n 1 abc123
```

Both of these commands will provide you with the necessary information about the specific commit you're interested in.

Output:



```
commit a9ba5a5a805fb0a25f5cb33b422f89fef641a0b7 (HEAD -> master, tag: v1.0, feature_branch)
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:36:13 2024 +0530

    Data Analysing
```

Experiment 10.

Analysing and Changing Git History

Write the command to list all commits made by the author "JohnDoe" between "2023- 01-01"and "2023-12-31."

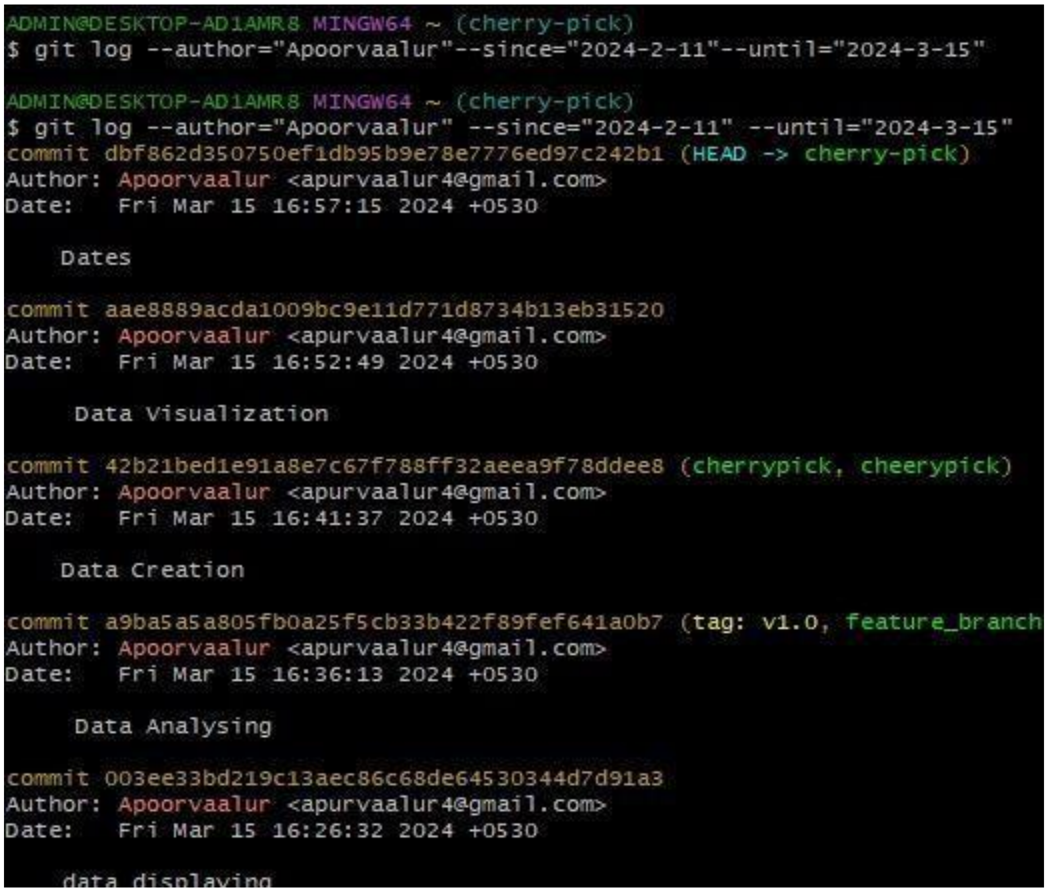
Solution:

To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31" in Git, you can use the git log command with the --author and --since and --until options. Here'sthe command:

```
$ git log --author="JohnDoe" --since="2023-01-01" --until="2023-12-31"
```

This command will display a list of commits made by the author "JohnDoe" that fall within thespecified date range, from January 1, 2023, to December 31, 2023. Make sure to adjust the author name and date range as needed for your specific use case.

Output:



```
ADMIN@DESKTOP-AD1AMR8 MINGW64 ~ (cherry-pick)
$ git log --author="Apoorvaalur"--since="2024-2-11"--until="2024-3-15"

ADMIN@DESKTOP-AD1AMR8 MINGW64 ~ (cherry-pick)
$ git log --author="Apoorvaalur" --since="2024-2-11" --until="2024-3-15"
commit dbf862d350750ef1db95b9e78e7776ed97c242b1 (HEAD -> cherry-pick)
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:57:15 2024 +0530

    Dates

commit aae8889acda1009bc9e11d771d8734b13eb31520
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:52:49 2024 +0530

     Data Visualization

commit 42b21bed1e91a8e7c67f788ff32aeea9f78ddee8 (cherrypick, cheerypick)
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:41:37 2024 +0530

    Data Creation

commit a9ba5a5a805fb0a25f5cb33b422f89fef641a0b7 (tag: v1.0, feature_branch
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:36:13 2024 +0530

     Data Analysing

commit 003ee33bd219c13aec86c68de64530344d7d91a3
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:26:32 2024 +0530

    data displaying
```

Experiment 11.

Analysing and Changing Git History

Write the command to display the last five commits in the repository's history.

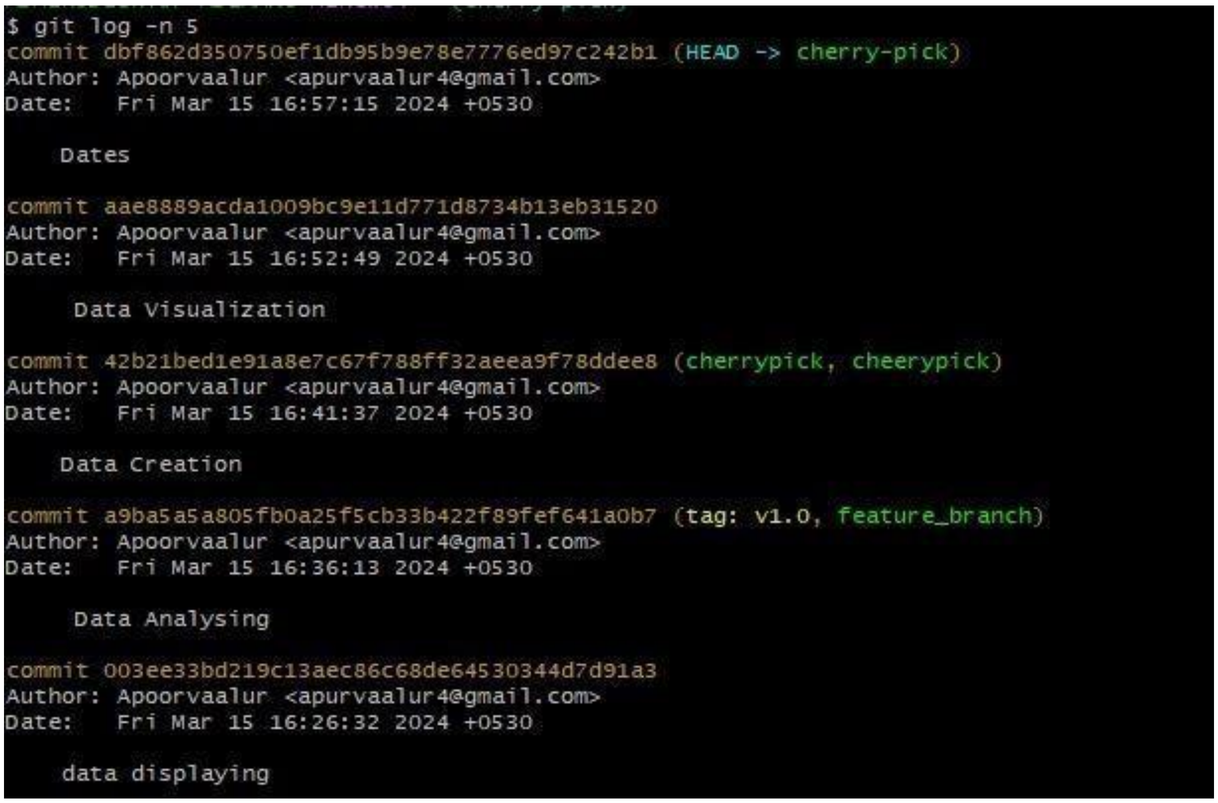
Solution:

To display the last five commits in a Git repository's history, you can use the git log command with the -n option, which limits the number of displayed commits. Here's the command:

```
$ git log -n 5
```

This command will show the last five commits in the repository's history. You can adjust thenumber after -n to display a different number of commits if needed.

Output:



```
$ git log -n 5
commit dbf862d350750ef1db95b9e78e7776ed97c242b1 (HEAD -> cherry-pick)
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:57:15 2024 +0530

    Dates

commit aae8889acda1009bc9e11d771d8734b13eb31520
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:52:49 2024 +0530

     Data Visualization

commit 42b21bed1e91a8e7c67f788ff32aeea9f78ddee8 (cherrypick, cheerypick)
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:41:37 2024 +0530

    Data Creation

commit a9ba5a5a805fb0a25f5cb33b422f89fef641a0b7 (tag: v1.0, feature_branch)
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:36:13 2024 +0530

     Data Analysing

commit 003ee33bd219c13aec86c68de64530344d7d91a3
Author: Apoorvaalur <apurvaalur4@gmail.com>
Date:   Fri Mar 15 16:26:32 2024 +0530

    data displaying
```

Experiment 12.

Analysing and Changing Git History
Write the command to undo the changes introduced by the commit with the ID "abc123".

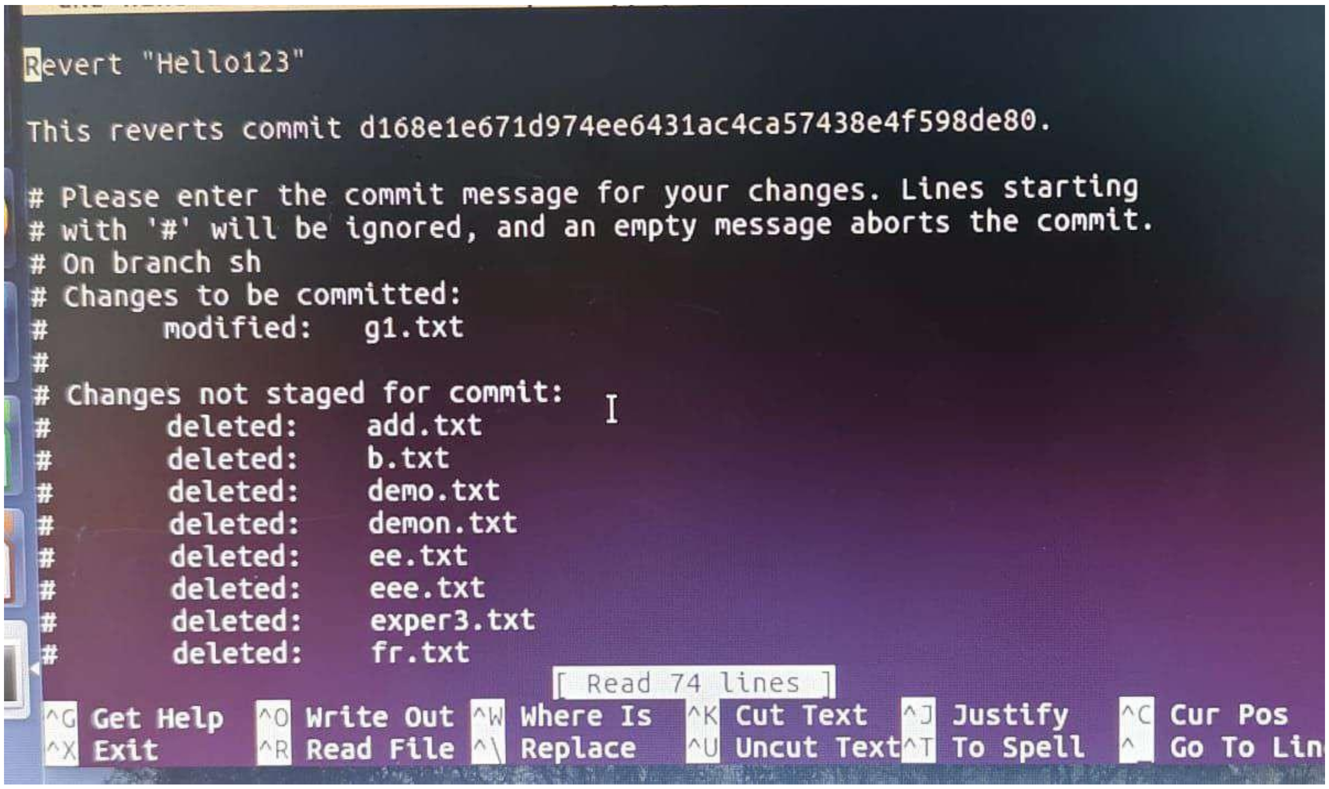
Solution:

To undo the changes introduced by a specific commit with the ID "abc123" in Git, you can use the git revert command. The git revert command creates a new commit that undoes the changes made by the specified commit, effectively "reverting" the commit. Here's the command:

```
$ git revert abc123
```

Replace "abc123" with the actual commit ID that you want to revert. After running this command, Git will create a new commit that negates the changes introduced by the specified commit. This is a safe way to undo changes in Git because it preserves the commit history andcreates a new commit to record the reversal of the changes.

Output:



```
Revert "Hello123"

This reverts commit d168e1e671d974ee6431ac4ca57438e4f598de80.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch sh
# Changes to be committed:
#       modified:   g1.txt
#
# Changes not staged for commit:
#       deleted:    add.txt
#       deleted:    b.txt
#       deleted:    demo.txt
#       deleted:    demon.txt
#       deleted:    ee.txt
#       deleted:    eee.txt
#       deleted:    exper3.txt
#       deleted:    fr.txt
                              [ Read 74 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^  Go To Lin
```