VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“Jnana Sangama”, Machhe, Belagavi, Karnataka-590018

Lab Experiment Record

Project Management with Git [BCSL358C]

Submitted in partial fulfillment towards AEC of 3rd semester of

Bachelor of Engineering

in

Computer Science and Engineering
(Artificial Intelligence & Machine Learning)

Submitted by

BASAVASIRI H L
4GW24CI006

DEPARTMENT OF CSE (Artificial Intelligence & Machine Learning)

GSSS INSTITUTE OF ENGINEERING & TECHNOLOGY FOR WOMEN

(Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi & Govt. of Karnataka)

K.R.S ROAD, METAGALLI, MYSURU-570016, KARNATAKA

(Accredited by NAAC)

2025-2026

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.

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

```
$ git add Text.md
```

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"
```

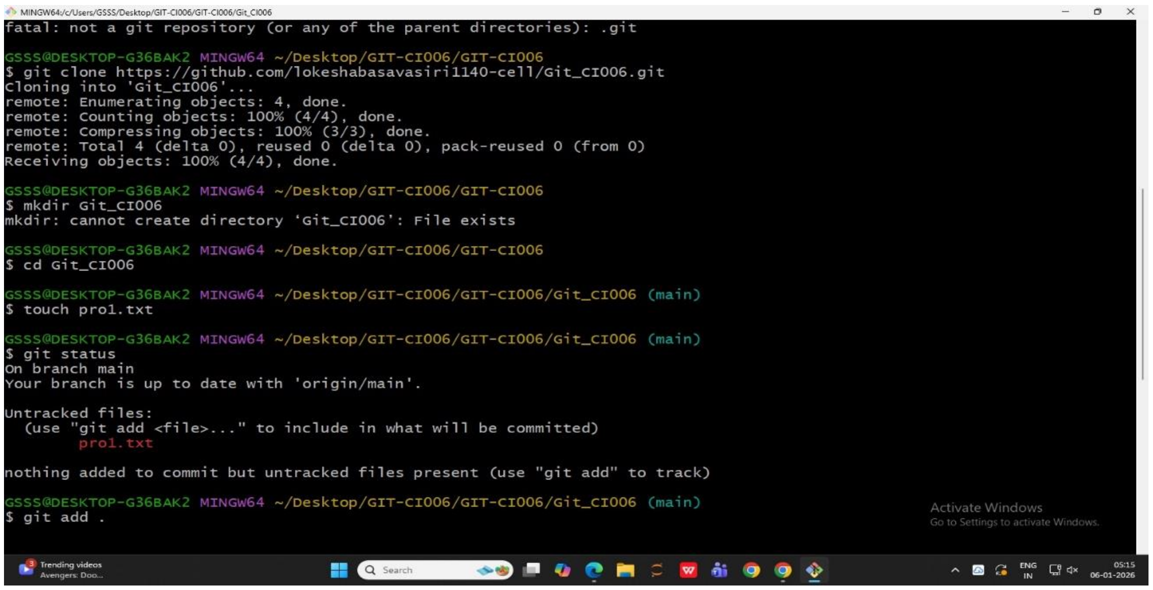
Your commit message should briefly describe the purpose or nature of the changes you made.

For example:

```
$ git commit -m "Add a new file called Text.md "
```

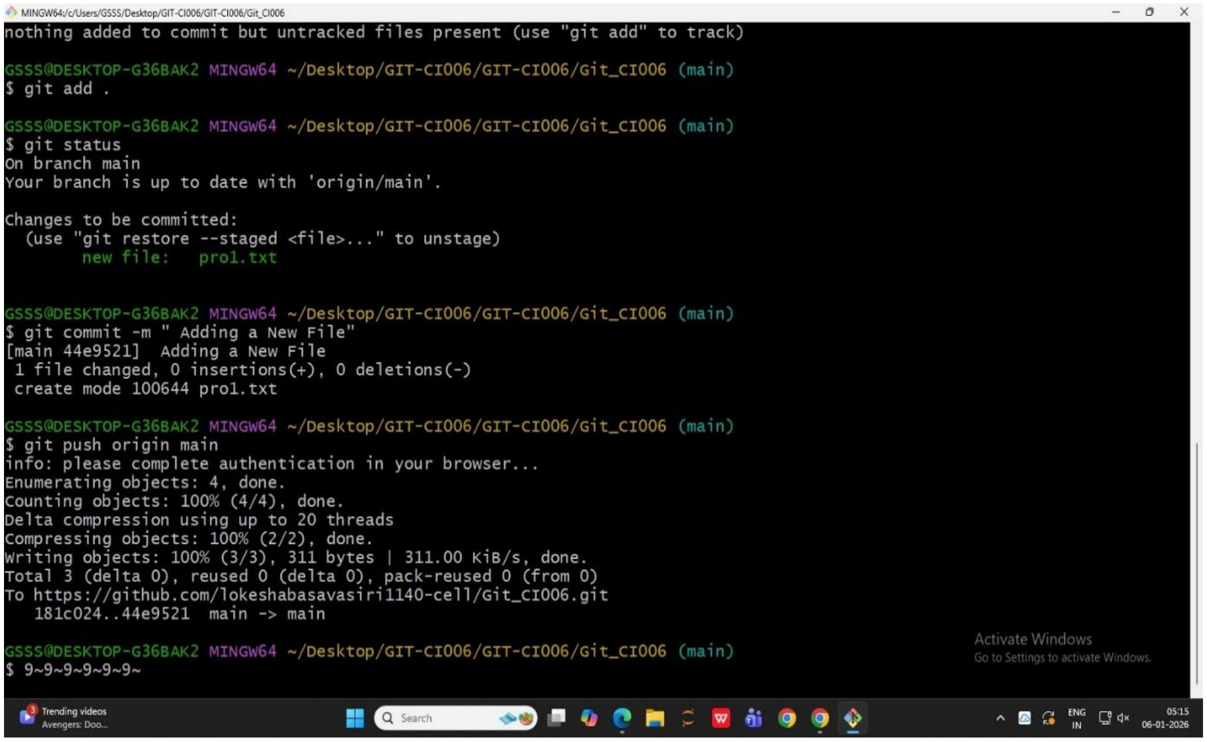
After these steps, your changes will be committed to the Git repository with the provided commit message. You now have a version of the repository with the new file and its history stored in Git.

URL : https://github.com/lokeshabasavasiri1140-cell/Git_CI006.git



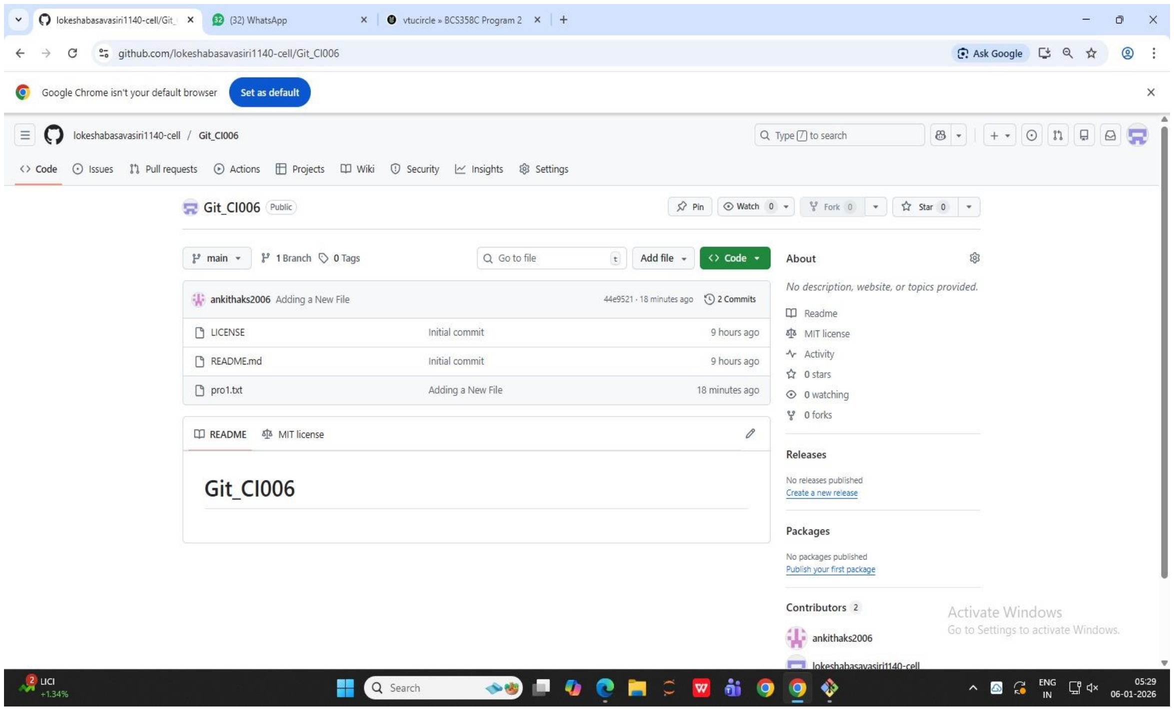
MINGW64:/c/Users/GSSS/Desktop/GIT-CI006/GIT-CI006/Git_CI006

```
fatal: not a git repository (or any of the parent directories): .git

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006
$ git clone https://github.com/lokeshabasavasiri1140-cell/Git_CI006.git
Cloning into 'Git_CI006'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (4/4), done.

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006
$ mkdir Git_CI006
mkdir: cannot create directory 'Git_CI006': File exists

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006
$ cd Git_CI006

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (main)
$ touch pro1.txt

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        pro1.txt

nothing added to commit but untracked files present (use "git add" to track)

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (main)
$ git add .
```



MINGW64:/c/Users/GSSS/Desktop/GIT-CI006/GIT-CI006/Git_CI006

```
nothing added to commit but untracked files present (use "git add" to track)

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (main)
$ git add .

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   pro1.txt


GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (main)
$ git commit -m " Adding a New File"
[main 44e9521]  Adding a New File
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 pro1.txt

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (main)
$ git push origin main
info: please complete authentication in your browser...
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 20 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 311 bytes | 311.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/lokeshabasavasiri1140-cell/Git_CI006.git
   181c024..44e9521  main -> main

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (main)
$ 9~9~9~9~9~9~
```



ankithaks2006 Adding a New File 44e9521 · 18 minutes ago 2 Commits

LICENSE	Initial commit	9 hours ago
README.md	Initial commit	9 hours ago
pro1.txt	Adding a New File	18 minutes ago

README MIT license

Git_CI006

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 as needed.

4. Stage and commit your changes in the "feature-branch":

```
$ git add .
$ git commit -m "Changed from master branch to feature-branch."
```

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

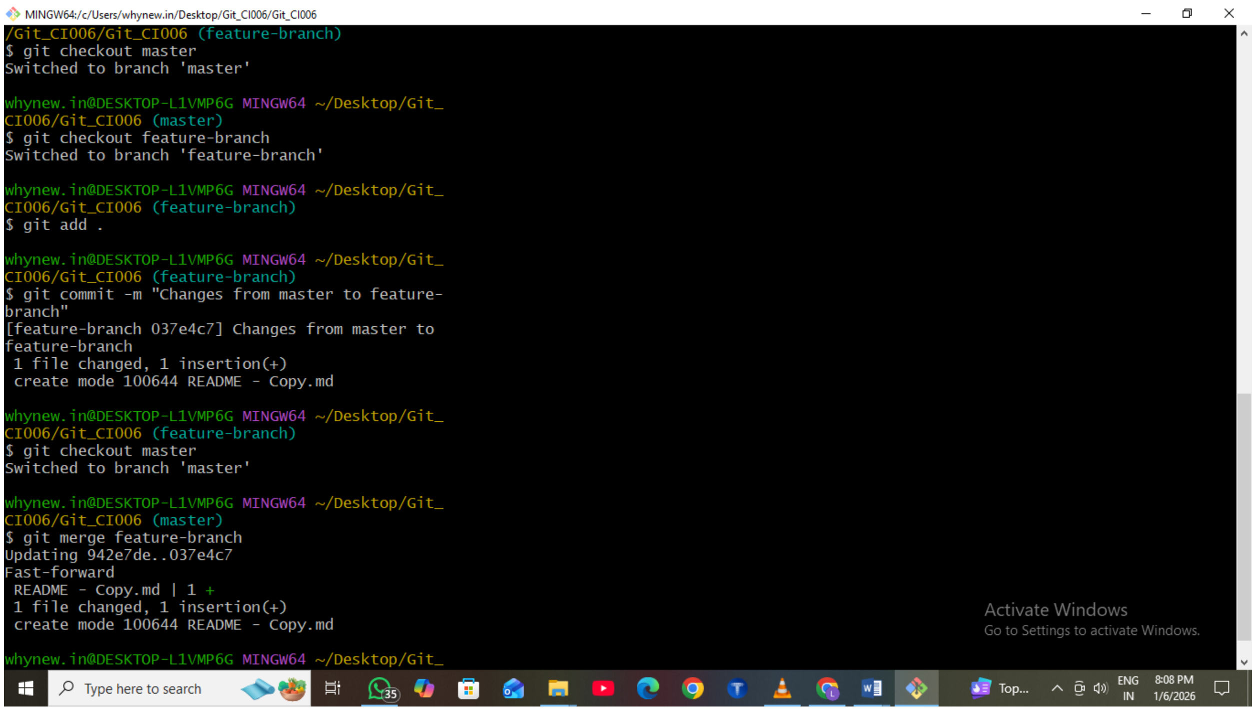
5. Switch back to the "master" branch:

```
$ 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. Your project's history will reflect the changes made in both branches.



```
/Git_CI006/Git_CI006 (feature-branch)
$ git checkout master
Switched to branch 'master'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_
CI006/Git_CI006 (master)
$ git checkout feature-branch
Switched to branch 'feature-branch'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_
CI006/Git_CI006 (feature-branch)
$ git add .

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_
CI006/Git_CI006 (feature-branch)
$ git commit -m "Changes from master to feature-
branch"
[feature-branch 037e4c7] Changes from master to
feature-branch
 1 file changed, 1 insertion(+)
 create mode 100644 README - Copy.md

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_
CI006/Git_CI006 (feature-branch)
$ git checkout master
Switched to branch 'master'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_
CI006/Git_CI006 (master)
$ git merge feature-branch
Updating 942e7de..037e4c7
Fast-forward
 README - Copy.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 README - Copy.md

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_
```

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 use the following commands:

1. Stash your changes:

$ git stash save "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:

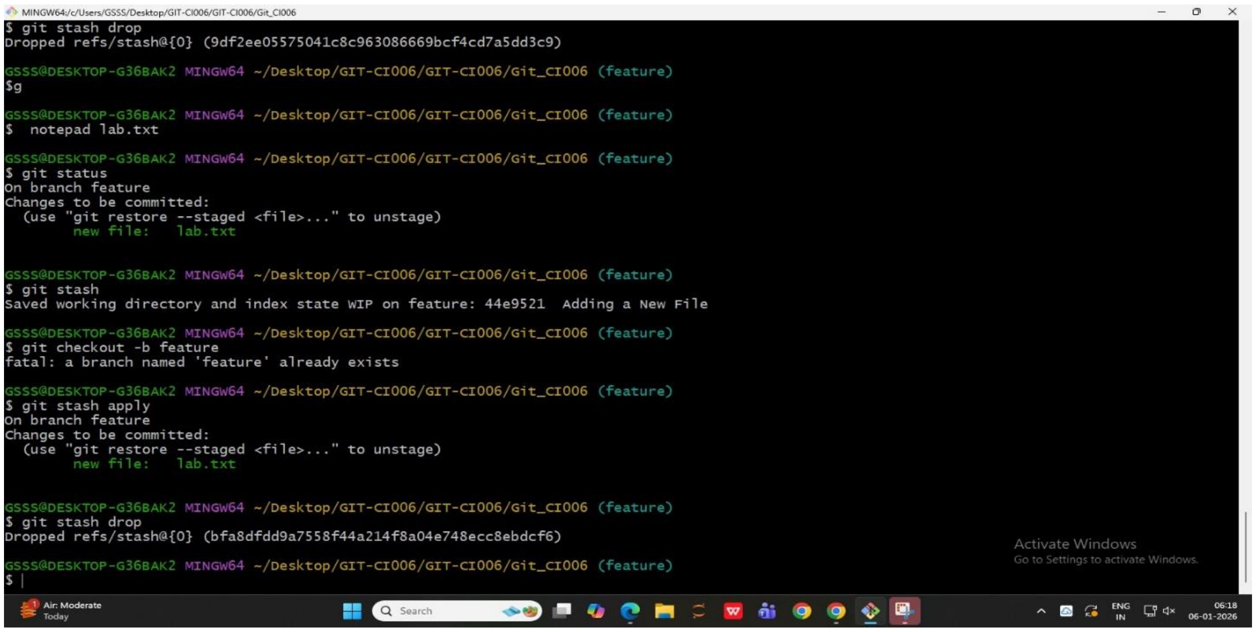
```
$ git checkout target-branch
```

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.



```
$ git stash drop
Dropped refs/stash@{0} (9df2ee05575041c8c963086669bcf4cd7a5dd3c9)

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (feature)
$g

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (feature)
$  notepad lab.txt

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (feature)
$ git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   lab.txt


GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (feature)
$ git stash
Saved working directory and index state WIP on feature: 44e9521  Adding a New File

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (feature)
$ git checkout -b feature
fatal: a branch named 'feature' already exists

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (feature)
$ git stash apply
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   lab.txt


GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (feature)
$ git stash drop
Dropped refs/stash@{0} (bfa8dfdd9a7558f44a214f8a04e748ecc8ebdcf6)

GSSS@DESKTOP-G36BAK2 MINGW64 ~/Desktop/GIT-CI006/GIT-CI006/Git_CI006 (feature)
$ |
```

MINGW64:/c/Users/GSSS/Desktop/GIT-CI006/GIT-CI006/Git_CI006
Activate Windows
Go to Settings to activate Windows.
Air: Moderate
Today
Search
ENG
IN
06:18
06-01-2026

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 were cloning a repository from GitHub, the URL might look like this:

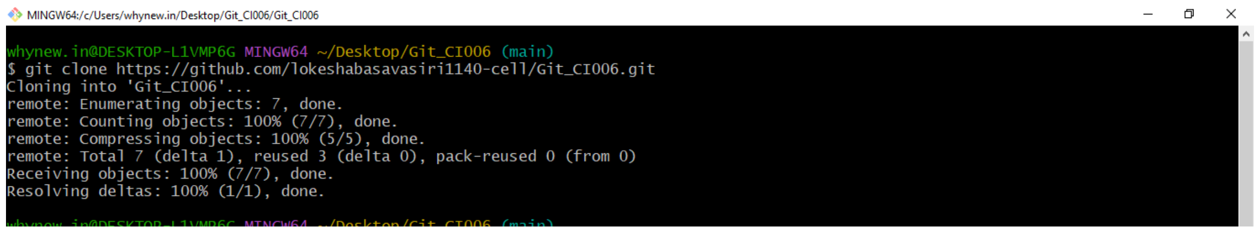
```
$ git clone <repository_url>
```

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 you want to clone.

4. Git will clone the repository to your local machine. Once the process is complete, you will 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 push those changes back to the remote repository as needed.


MINGW64:/c/Users/whynew.in/Desktop/Git_CI006/Git_CI006

```
whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006 (main)
$ git clone https://github.com/lokeshabasavasiri1140-cell/Git_CI006.git
Cloning into 'Git_CI006'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 1), reused 3 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (7/7), done.
Resolving deltas: 100% (1/1), 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, replace origin 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 notify you 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:

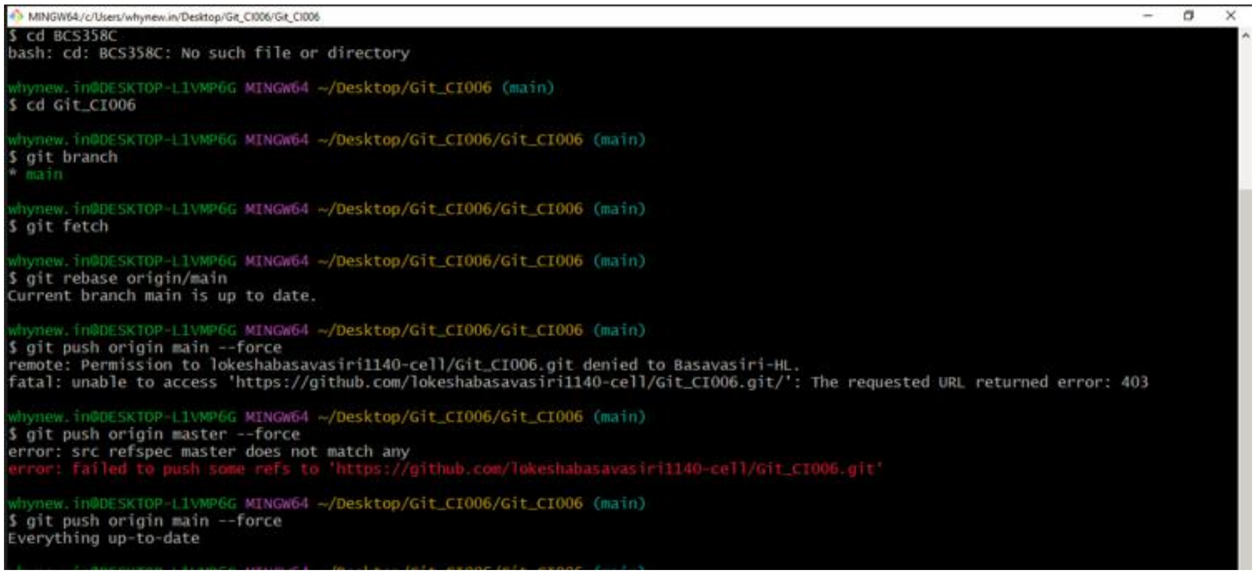
```
$ git rebase –continue
```

6. After resolving any conflicts and completing the rebase, you have successfully updated your 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 other developers' changes:

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

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



```
$ cd BCS358C
bash: cd: BCS358C: No such file or directory

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006 (main)
$ cd Git_CI006

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git branch
* main

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git fetch

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git rebase origin/main
Current branch main is up to date.

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git push origin main --force
remote: Permission to lokeshabasavasiri1140-cell/Git_CI006.git denied to Basavasiri-HL.
fatal: unable to access 'https://github.com/lokeshabasavasiri1140-cell/Git_CI006.git/': The requested URL returned error: 403

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git push origin master --force
error: src refspec master does not match any
error: failed to push some refs to 'https://github.com/lokeshabasavasiri1140-cell/Git_CI006.git'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git push origin main --force
Everything up-to-date
```

Experiment 6: Creating and Managing Branches:

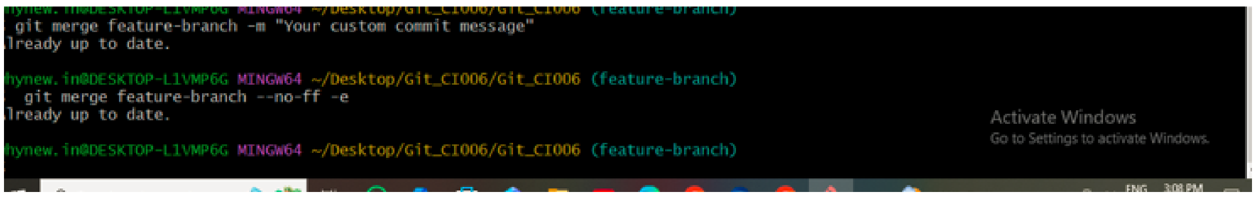
Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

Solution:

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

```
$ git checkout master
$ git merge feature-branch -m "feature-branch merged."
```

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 created when you merge "feature-branch" into "master."



```
hynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (feature-branch)
 git merge feature-branch -m "Your custom commit message"
lready up to date.

hynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (feature-branch)
  git merge feature-branch --no-ff -e
lready up to date.

hynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (feature-branch)
```

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 specific commit in your local repository, you can use the following command

```
GIT TAG V1.0 <COMMIT_HASH>
```

Replace **<commit_hash>** with the actual hash of the commit for which you want to create the tag.

For example, if you want to tag the latest commit, you can use the following:

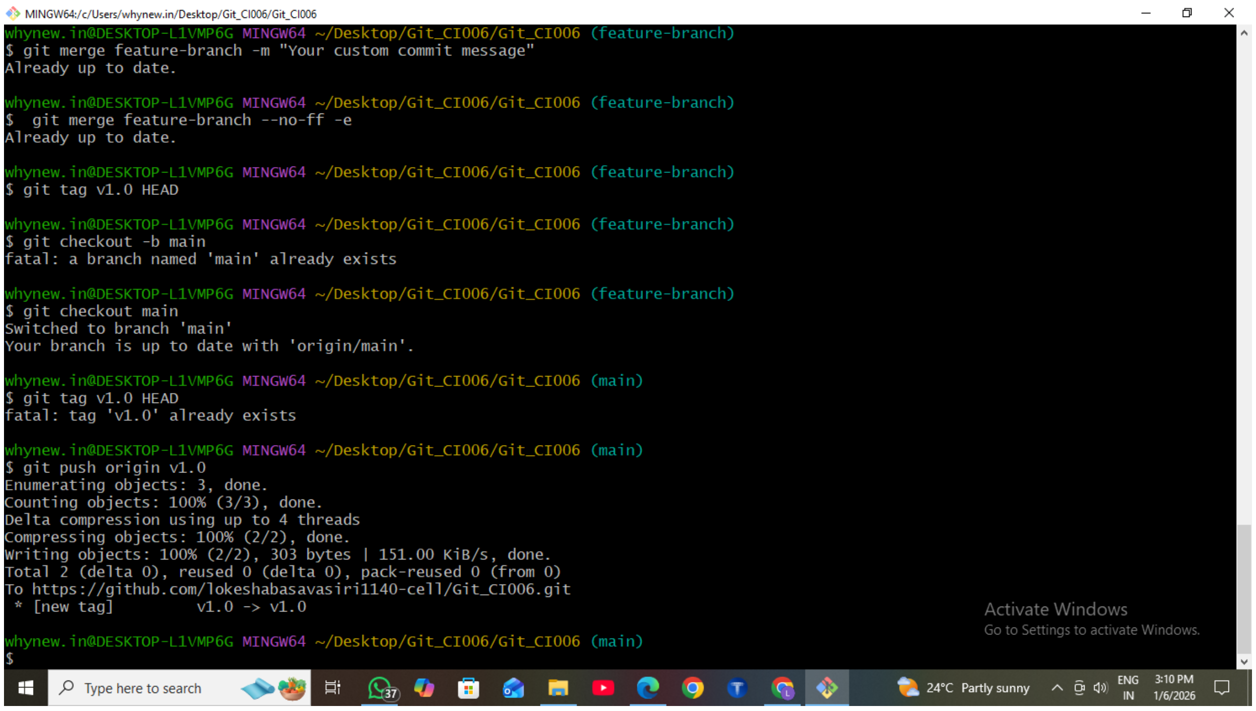
```
GIT TAG V1.0  HEAD
```

This creates a lightweight tag pointing to the specified commit. Lightweight tags are simply pointers to specific commits and contain only the commit checksum.

If you want to push the tag to a remote repository, you can use:

```
GIT PUSH ORIGIN V1.0
```

This command pushes the tag named “v1.0” to to the remote repository. Keep in mind that Git tags, by default, are not automatically pushed to remotes, so you need to explicitly push them if needed.


MINGW64:/c/Users/whynew.in/Desktop/Git_CI006/Git_CI006
Type here to search
24°C Partly sunny
ENG IN
3:10 PM 1/6/2026
Activate Windows
Go to Settings to activate Windows.

```
whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (feature-branch)
$ git merge feature-branch -m "Your custom commit message"
Already up to date.

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (feature-branch)
$  git merge feature-branch --no-ff -e
Already up to date.

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (feature-branch)
$ git tag v1.0 HEAD

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (feature-branch)
$ git checkout -b main
fatal: a branch named 'main' already exists

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (feature-branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git tag v1.0 HEAD
fatal: tag 'v1.0' already exists

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git push origin v1.0
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 303 bytes | 151.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/lokeshabasavasiri1140-cell/Git_CI006.git
 * [new tag]         v1.0 -> v1.0

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$
```

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-comit>

Replace `<start-commit>` and `<end-commit>` with the commit hashes or references that define the range of commits you want to cherry-pick. The ^ (caret) symbol is used to exclude the starting commit itself from the range.

For example, if you want to cherry-pick the commits from commit A to commit B (excluding A) from “source-branch” to the current branch, you would run:

git cherry-pick A^..B

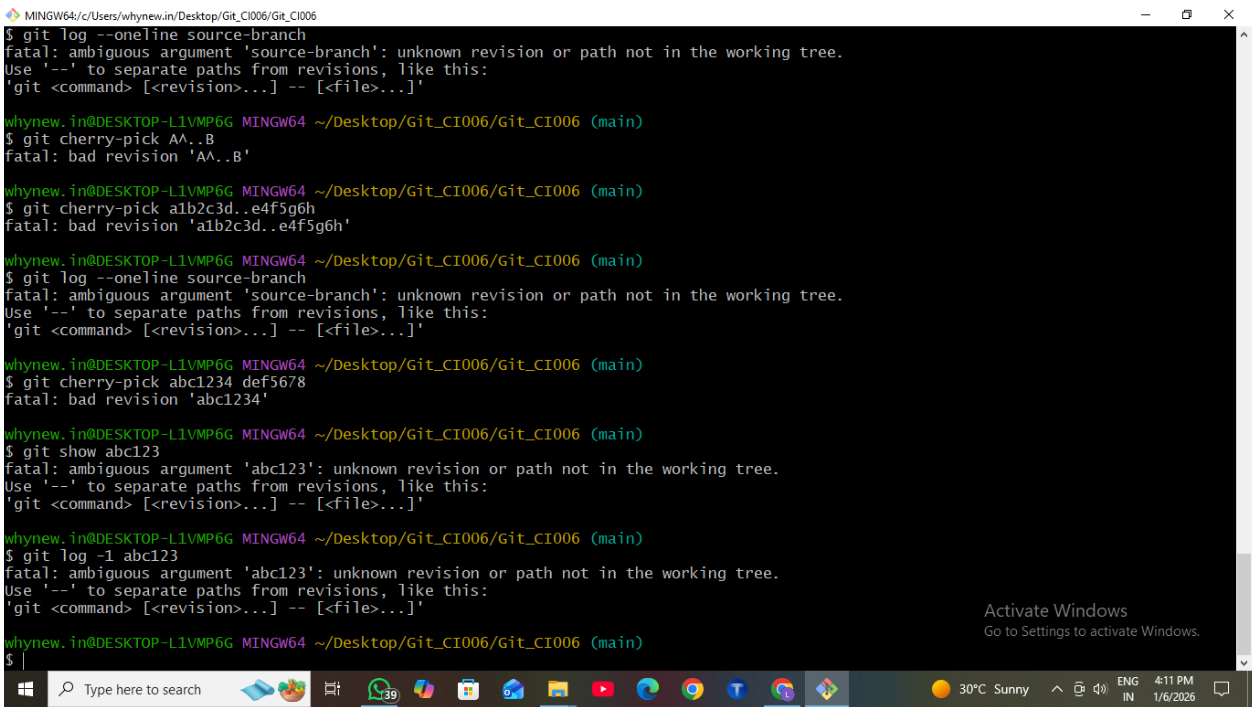
After running this command, Git will apply the specified range of commits onto your current branch. If there are any conflicts, Git will pause the cherry-pick process and ask you to resolve them. After resolving conflicts, you can continue the cherry-pick with:

git cherry-pick --continue

If you encounter issues and need to abort the cherry-pick operation, you can use:

git cherry-pick --abort

Remember that cherry-picking introduces new commits based on the changes from the source branch, so conflicts may arise, and manual intervention might be required.



```
$ git log --oneline source-branch
fatal: ambiguous argument 'source-branch': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git cherry-pick A^..B
fatal: bad revision 'A^..B'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git cherry-pick a1b2c3d..e4f5g6h
fatal: bad revision 'a1b2c3d..e4f5g6h'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git log --oneline source-branch
fatal: ambiguous argument 'source-branch': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git cherry-pick abc1234 def5678
fatal: bad revision 'abc1234'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git show abc123
fatal: ambiguous argument 'abc123': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ git log -1 abc123
fatal: ambiguous argument 'abc123': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006/Git_CI006 (main)
$ |
```

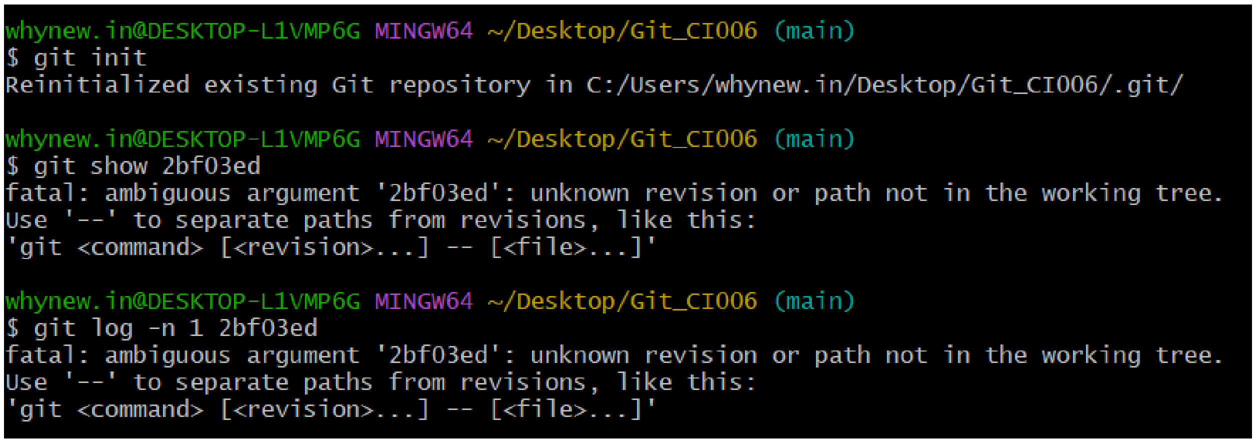
Experiment 9: Analysing and Changing Git History:

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, you can 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, and the changes introduced by that commit.
For example:
$ git show 2bf03ed
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 2bf03ed
Both of these commands will provide you with the necessary information about the specific commit you're interested in.



```
whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006 (main)
$ git init
Reinitialized existing Git repository in C:/Users/whynew.in/Desktop/Git_CI006/.git/

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006 (main)
$ git show 2bf03ed
fatal: ambiguous argument '2bf03ed': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006 (main)
$ git log -n 1 2bf03ed
fatal: ambiguous argument '2bf03ed': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'
```

Experiment 10: Analysing and Changing Git History:

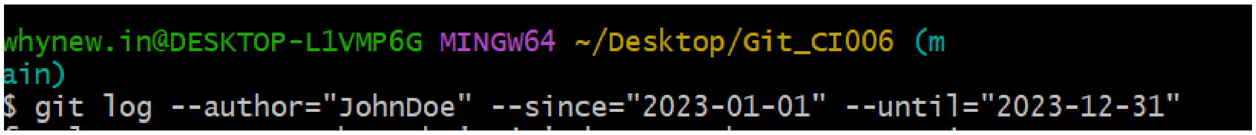
Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."

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's the 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 the specified 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.



```
whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006 (main)
$ git log --author="JohnDoe" --since="2023-01-01" --until="2023-12-31"
```

No output occurs when the author name or date range does not match any commits in the repository.

Experiment 11: Analysing and Changing Git History:

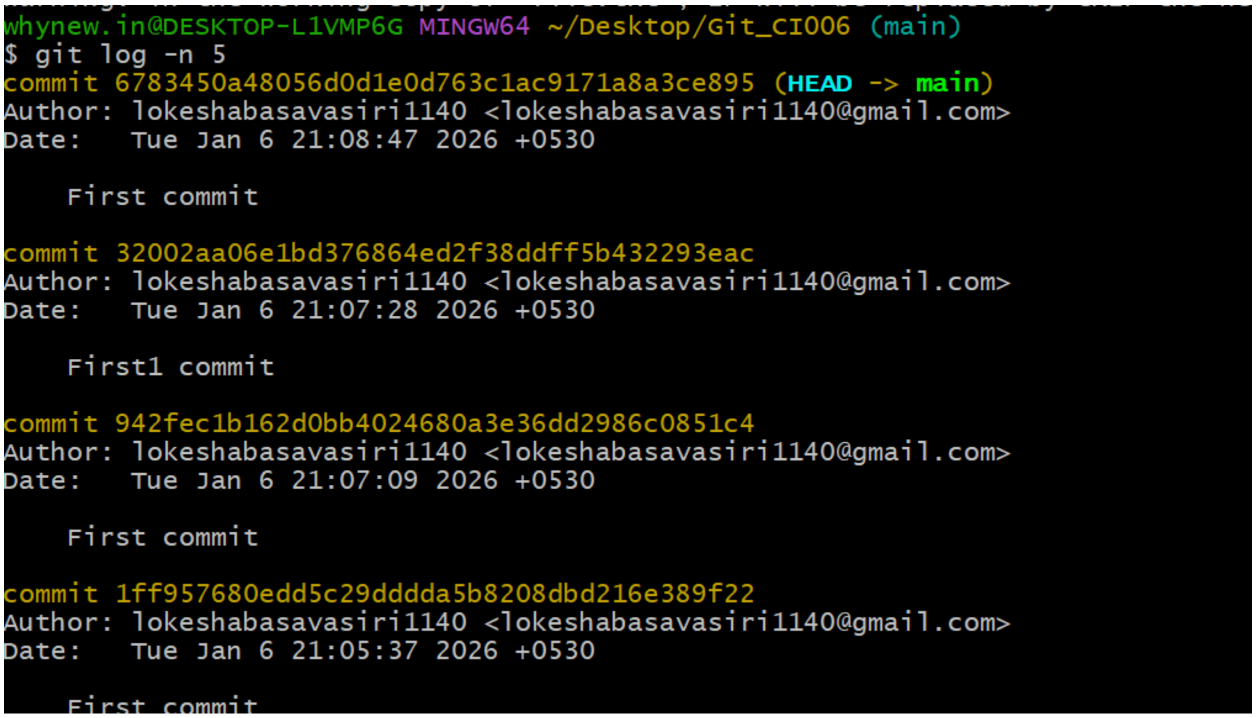
Write the command to display the last five commits in the repository's history.

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 the number after -n to display a different number of commits if needed.



```
whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop/Git_CI006 (main)
$ git log -n 5
commit 6783450a48056d0d1e0d763c1ac9171a8a3ce895 (HEAD -> main)
Author: lokeshabasavasiri1140 <lokeshabasavasiri1140@gmail.com>
Date:   Tue Jan 6 21:08:47 2026 +0530

    First commit

commit 32002aa06e1bd376864ed2f38ddff5b432293eac
Author: lokeshabasavasiri1140 <lokeshabasavasiri1140@gmail.com>
Date:   Tue Jan 6 21:07:28 2026 +0530

    First1 commit

commit 942fec1b162d0bb4024680a3e36dd2986c0851c4
Author: lokeshabasavasiri1140 <lokeshabasavasiri1140@gmail.com>
Date:   Tue Jan 6 21:07:09 2026 +0530

    First commit

commit 1ff957680edd5c29dddda5b8208dbd216e389f22
Author: lokeshabasavasiri1140 <lokeshabasavasiri1140@gmail.com>
Date:   Tue Jan 6 21:05:37 2026 +0530

    First commit
```

Experiment 12: Analysing and Changing Git History:

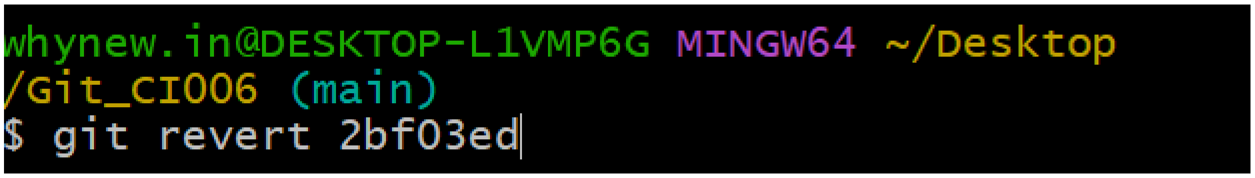
Write the command to undo the changes introduced by the commit with the ID "abc123".

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 and creates a new commit to record the reversal of the changes.



```
whynew.in@DESKTOP-L1VMP6G MINGW64 ~/Desktop
/Git_CI006 (main)
$ git revert 2bf03ed
```