

**Q1. 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.**

1. Initialize a new Git repository:

```
mkdir project_directory
```

```
cd project_directory
```

```
git init
```

2. Create a new file:

```
File> new> your_new_file.txt
```

3. Add the file to the staging area:

```
git add your_new_file.txt
```

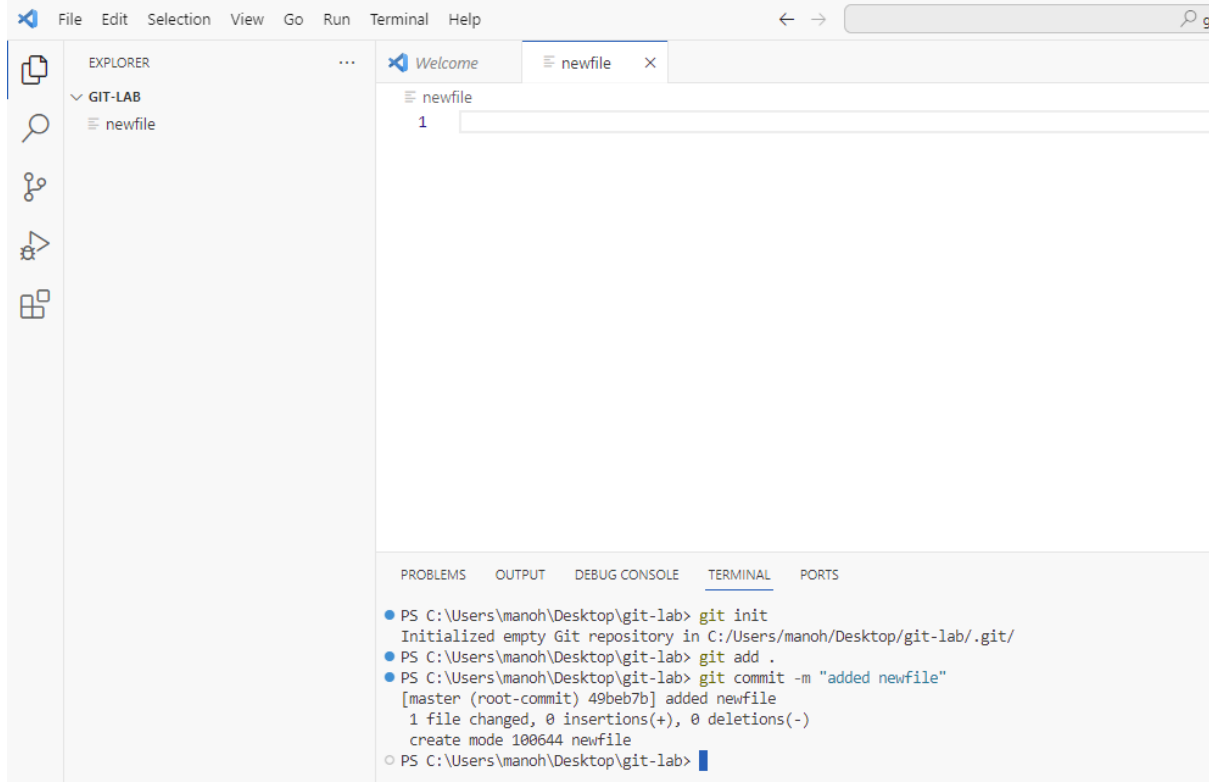
```
git add . (to add multiple files to staging area)
```

If you want to add all new and modified files to the staging area, you can use:

```
git add .
```

4. Commit the changes with a message:

```
git commit -m "Initial commit added file"
```



## Q2. Creating and Managing Branches Create a new branch named "feature- branch." Switch to the "master" branch. Merge the "feature-branch" into "master."

1. Initialize a new Git repository:

```
mkdir project_directory
```

```
cd project_directory
```

```
git init
```

2. Create a new file:

```
File> new> your_new_file.txt
```

3. Add the file to the staging area:

```
git add your_new_file.txt
```

```
git add . (to add multiple files to staging area)
```

If you want to add all new and modified files to the staging area, you can use:

```
git add .
```

4. Commit the changes with a message:

```
git commit -m "Initial commit added file"
```

Repeat above steps multiple times to commit multiple files on a master branch..

5. Create a new branch named "feature- branch":

```
git branch feature-branch
```

Alternatively, you can create and switch to the new branch in one step:

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

New branch called feature-branch will be created and it has backup of all files of master branch.

In feature-branch we can add and test new features on the project, once the feature program files are tested and verified we can merge the feature committed files on to master branch by below mentioned steps.

6. Switch back to the "master" branch:

```
git checkout master
```

7. Merge "feature-branch" into "master":

```
git merge feature-branch
```

If there are no conflicts, Git will automatically perform a fast-forward merge. If there are conflicts, Git will prompt you to resolve them before completing the merge. If you used git checkout -b feature branch to create and switch to the new branch, you can switch back to master and merge in a single command:

```
git checkout master git merge feature-branch
```

Now, the changes from "feature-branch" are merged into the "master" branch. If you no longer need the "feature-branch," you can delete it:

```
git branch -d feature-branch
```

This assumes that the changes in "feature- branch" do not conflict with changes in the "master" branch. If conflicts arise during the merge, you'll need to resolve them manually before completing the merge.

### **Q3. Creating and Managing Branches Write the commands to stash your changes, switch branches, and then apply the stashed changes.**

1. Stash your changes:

```
git add .
```

```
git stash -m "added search file to stash"
```

Saved working directory and index state On master: added search file to stash

```
git add .
```

```
git stash -m "added sort file to stash"
```

Saved working directory and index state On master: added sort file to stash

git add .

git stash -m "added arrays file to stash"

Saved working directory and index state On master: added arrays file to stash

git stash list

stash@{0}: On master: added arrays file to stash

stash@{1}: On master: added sort file to stash

stash@{2}: On master: added search file to stash

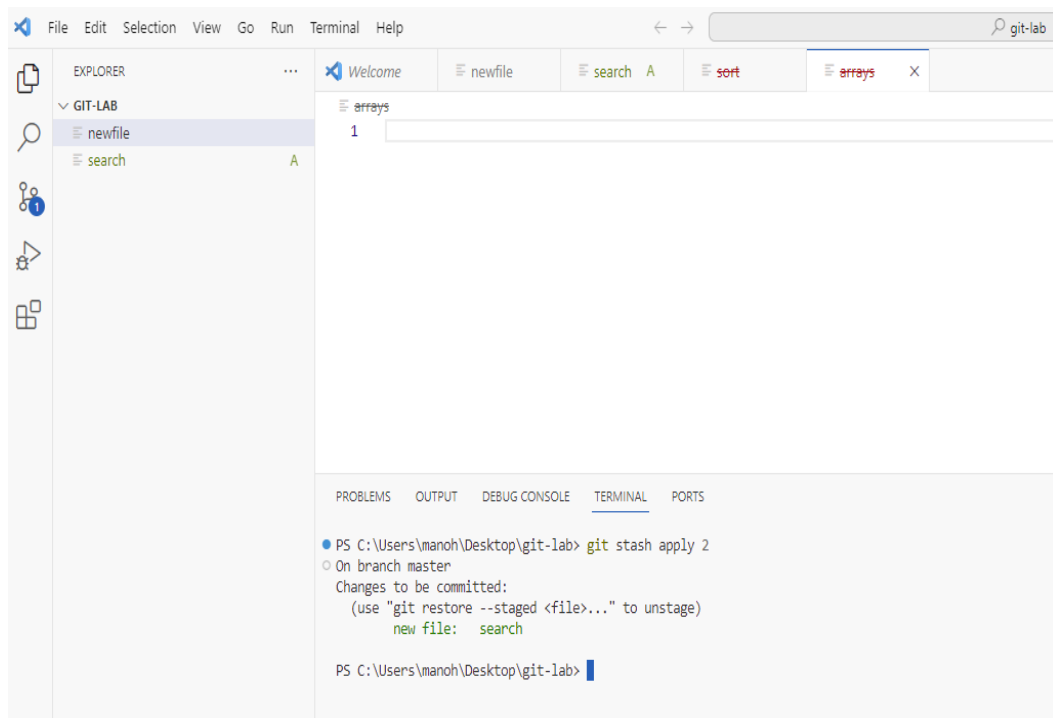
git stash apply 2

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: search



This command will save your local changes in a temporary area, allowing you to switch branches without committing the changes.

2. Switch to another branch:

git checkout your-desired-branch

3. Apply the stashed changes:

```
git stash apply
```

If you have multiple stashes and want to apply a specific stash, you can use:

```
git stash apply 1
```

After applying the stash, your changes are reapplied to the working directory.

4. Remove the applied stash (optional):

If you no longer need the stash after applying it, you can remove it:

```
git stash drop
```

To remove a specific stash:

```
git stash drop 2
```

If you want to apply and drop in one step, you can use git stash pop:

```
git stash pop
```

Now, you've successfully stashed your changes, switched branches, and applied the stashed changes.

#### Q4. Collaboration and Remote Repositories

Clone a remote Git repository to your local machine. To clone a remote Git repository to your local machine, you can use the git clone command. Here's the general syntax:

```
git clone <repository_url>
```

Replace <repository\_url> with the actual URL of the Git repository you want to clone. For example:

```
git clone https://github.com/example
```

```
git clone https://github.com/manoharnelli7/gitlab.git
```

## PROJECT MANAGEMENT WITH GIT MANUAL (BCS358C) 3<sup>rd</sup> SEMESTER CSE 2023-24

### Cloning into 'gitlab'...

remote: Enumerating objects: 46, done.

remote: Counting objects: 100% (46/46), done.

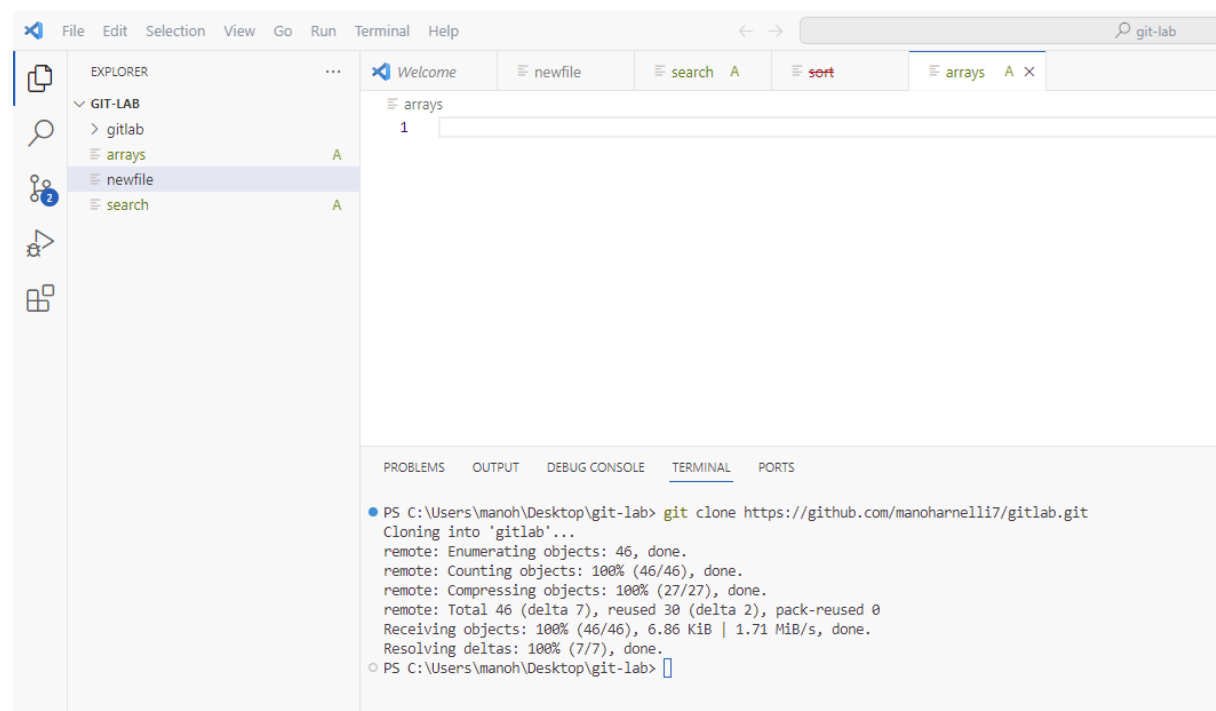
remote: Compressing objects: 100% (27/27), done.

remote: Total 46 (delta 7), reused 30 (delta 2), pack-reused 0

Receiving objects: 100% (46/46), 6.86 KiB | 1.71 MiB/s, done.

Resolving deltas: 100% (7/7), done.

This command will create a new directory with the name of the repository and download all the files from the remote repository into that directory. If the repository is private and requires authentication, you might need to use the SSH URL or provide your credentials during the cloning process.



After running the git clone command, you'll have a local copy of the remote repository on your machine, and you can start working with the code.

### Q5. Collaboration and Remote Repositories Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

1. Fetch the latest changes from the remote repository:

First clone from the remote repository

Clone a remote Git repository to your local machine. To clone a remote Git repository to your local machine, you can use the git clone command. Here's the general syntax:

**git clone <repository\_url>**

Replace <repository\_url> with the actual URL of the Git repository you want to clone. For example:

*git clone https://github.com/manoharnelli7/gitlab.git*

2. Make changes in remote repository , then fetch the changes into local repository using the command

*git pull <remote repository address>*

*git pull https://github.com/manoharnelli7/gitlab.git*

This command fetches the latest changes from the remote repository without automatically merging them into your local branches.

## **2. Rebase your local branch onto the updated remote branch:**

1. Initialize a new Git repository:

```
mkdir project_directory
```

```
cd project_directory
```

```
git init
```

2. Create a new file:

```
File> new> your_new_file.txt
```

3. Add the file to the staging area:

```
git add your_new_file.txt
```

```
git add . (to add multiple files to staging area)
```

If you want to add all new and modified files to the staging area, you can use:

```
git add .
```

4. Commit the changes with a message:

```
git commit -m "Initial commit added file"
```

Repeat above steps multiple times to commit multiple files on a master branch..

5. Create a new branch named "feature- branch":

```
git branch feature-branch
```

Alternatively, you can create and switch to the new branch in one step:

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

New branch called feature-branch will be created and it has backup of all files of master branch.

In feature-branch we can add and test new features on the project, once the feature program files are tested and verified we can merge the feature committed files on to master branch by below mentioned steps.

6. Switch back to the "master" branch:

`git checkout master`

7. Rebase "feature-branch" into "master":

`git rebase feature-branch`

Assuming you are currently on the branch you want to update (replace your-branch with the actual name of your branch):

...

`git rebase origin/your-branch`

This command applies your local commits on top of the changes fetched from the remote branch. If conflicts arise, Git will pause the rebase process and ask you to resolve them. Alternatively, you can use the interactive rebase to review and modify commits during the rebase:

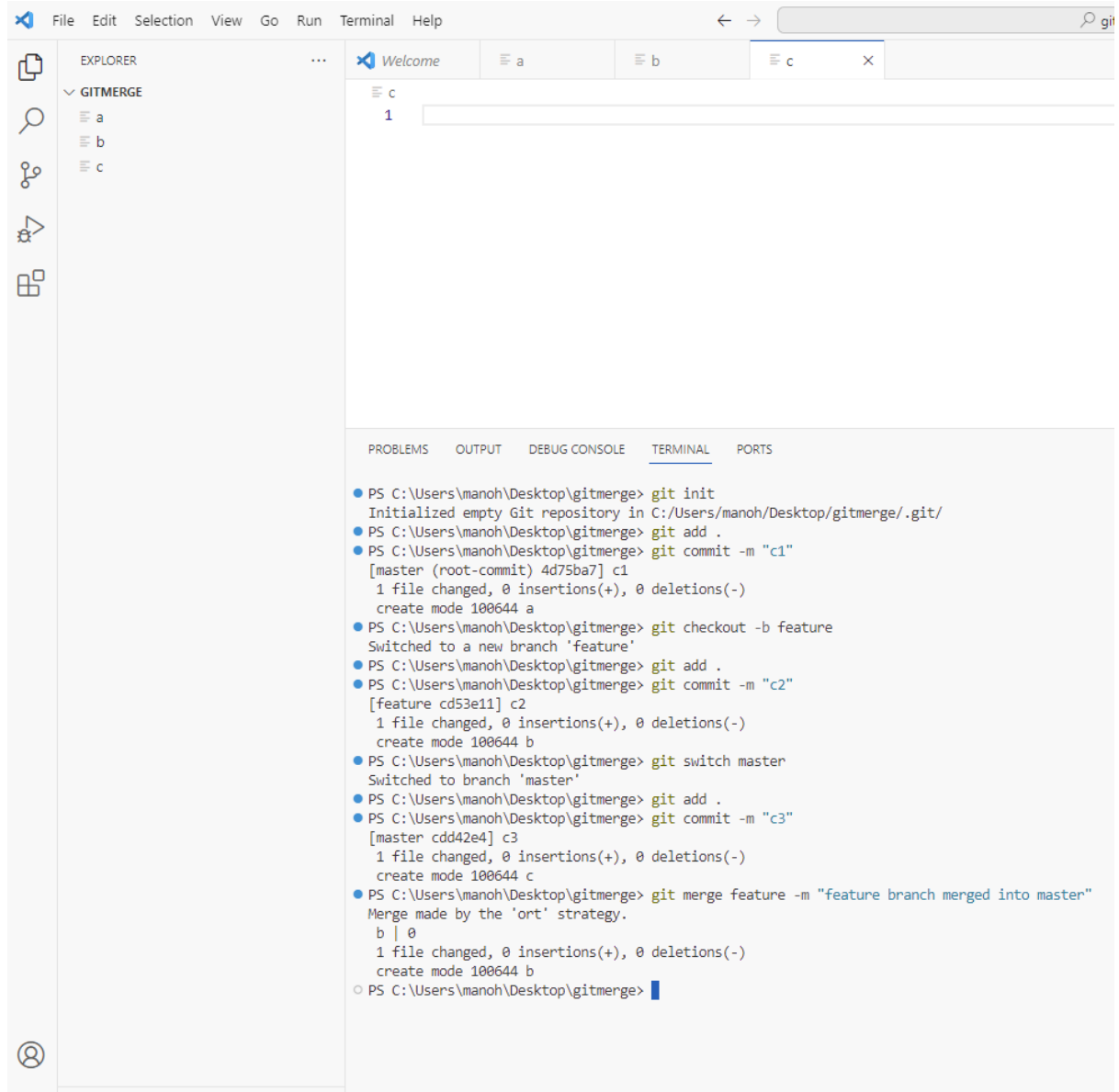
**Q6. Collaboration and Remote Repositories Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge. To merge "feature-branch" into "master" and provide a custom commit message, you can use the following command:**

`git merge feature-branch -m "Your custom commit message"`

Replace "Your custom commit message" with the actual message you want to use for the merge commit. This command performs the merge and creates a new commit on the "master" branch with the specified message. If there are no conflicts, Git will complete the merge automatically. If conflicts occur during the merge, Git will pause and prompt you to resolve the conflicts manually.



## PROJECT MANAGEMENT WITH GIT MANUAL (BCS358C) 3<sup>rd</sup> SEMESTER CSE 2023-24



git init

Initialized empty Git repository in C:/Users/manoh/Desktop/gitmerge/.git/

git add .

git commit -m "c1"

[master (root-commit) 4d75ba7] c1

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 a

git checkout -b feature

Switched to a new branch 'feature'

git add .

git commit -m "c2"

[feature cd53e11] c2

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 b

git switch master

Switched to branch 'master'

git add .

git commit -m "c3"

[master cdd42e4] c3

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 c

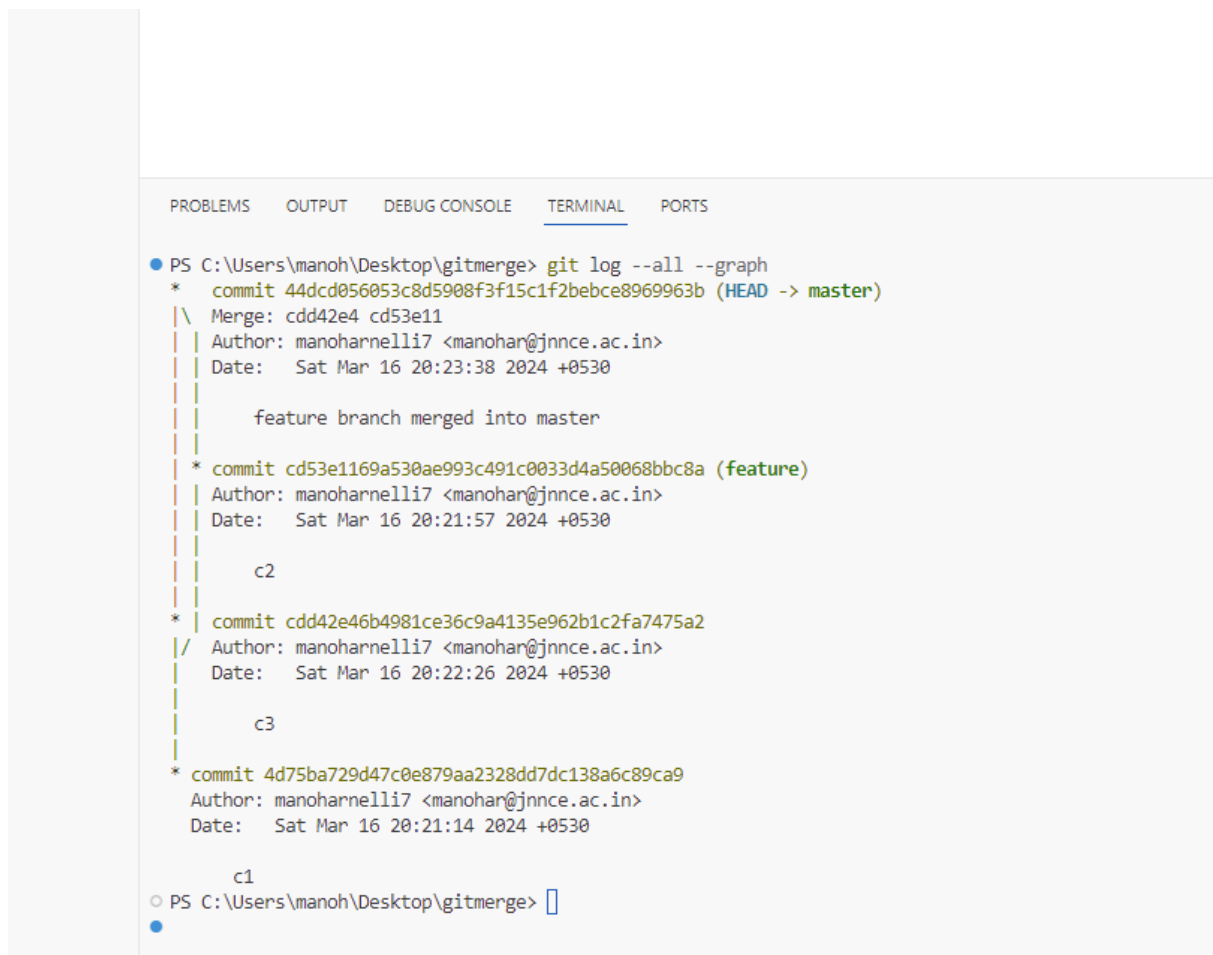
git merge feature -m "feature branch merged into master"

Merge made by the 'ort' strategy.

b | 0

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 b



```
PS C:\Users\manoh\Desktop\gitmerge> git log --all --graph
* commit 44dcd056053c8d5908f3f15c1f2bebc8969963b (HEAD -> master)
  | Merge: cdd42e4 cd53e11
  | Author: manoharnelli7 <manohar@jnnce.ac.in>
  | Date: Sat Mar 16 20:23:38 2024 +0530
  |
  | feature branch merged into master
  |
  * commit cd53e1169a530ae993c491c0033d4a50068bbc8a (feature)
  | Author: manoharnelli7 <manohar@jnnce.ac.in>
  | Date: Sat Mar 16 20:21:57 2024 +0530
  |
  | c2
  |
  * commit cdd42e46b4981ce36c9a4135e962b1c2fa7475a2
  |/ Author: manoharnelli7 <manohar@jnnce.ac.in>
  | Date: Sat Mar 16 20:22:26 2024 +0530
  |
  | c3
  |
  * commit 4d75ba729d47c0e879aa2328dd7dc138a6c89ca9
  | Author: manoharnelli7 <manohar@jnnce.ac.in>
  | Date: Sat Mar 16 20:21:14 2024 +0530
  |
  | c1
PS C:\Users\manoh\Desktop\gitmerge>
```

## Q7. Git Tags and Releases

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

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 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.

#### Step 1:

Checkout the branch where you want to create the tag

```
git checkout "branch name"
```

example : git checkout master

---

#### Step 2:

Create tag with some name

`git tag "tag name"`

example : `git tag v1.0`

`git tag -a v1.0 -m "ver 1 of .." (to create annotated tags)`

---

Step 3:

Display or Show tags

`git tag`

`git show v1.0`

`git tag -l "v1.*"`

---

Step 4:

Push tags to remote

`git push origin v1.0`

`git push origin --tags`

`git push --tags`

(to push all tags at once)

---

Step 5:

Delete tags (if required only)

to delete tags from local :

`git tag -d v1.0`

`git tag --delete v1.0`

to delete tags from remote :

`git push origin -d v1.0`

`git push origin --delete v1.0`

`git push origin :v1.0`

to delete multiple tags at once:

`git tag -d v1.0 v1.1 (local)`

`git push origin -d v1.0 v1.1 (remote)`

### Checking out TAGS

We cannot checkout tags in git

We can create a branch from a tag and checkout the branch

git checkout -b "branch name" "tag name"

example : git checkout -b ReleaseVer1 v1.0

---

### Creating TAGS from past commits

git tag "tag name" "reference of commit"

example : git tag v1.2 5fdb03

## Q8. Advanced Git Operations

**Write the command to cherry-pick a range of commits from "source-branch" to the current branch.**

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

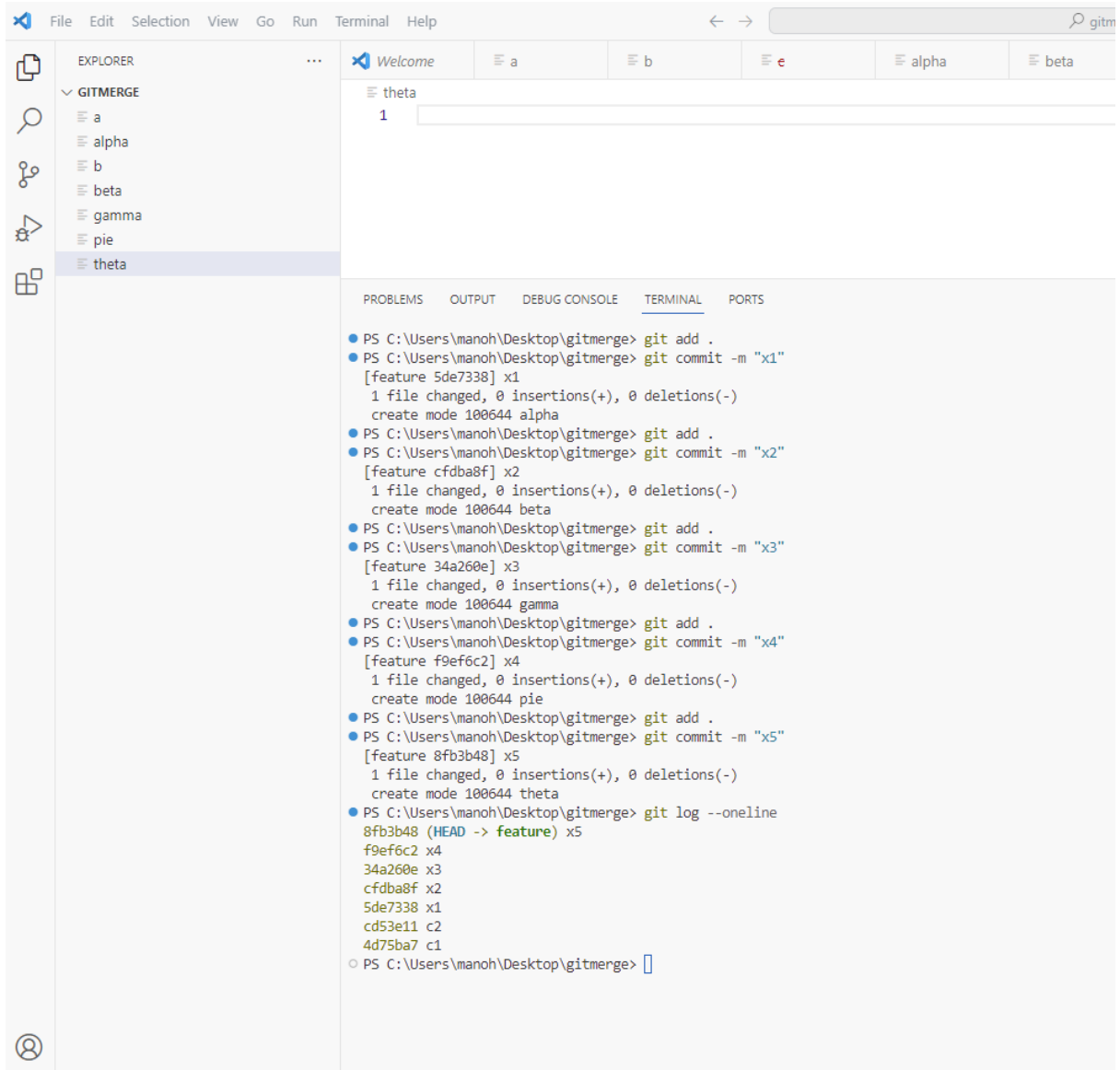
**git cherry-pick <start-commit>^..<end-commit>**

Replace <start-commit> 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:



```

PS C:\Users\manoh\Desktop\gitmerge> git add .
PS C:\Users\manoh\Desktop\gitmerge> git commit -m "x1"
[feature 5de7338] x1
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 alpha
PS C:\Users\manoh\Desktop\gitmerge> git add .
PS C:\Users\manoh\Desktop\gitmerge> git commit -m "x2"
[feature cfdb8f] x2
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 beta
PS C:\Users\manoh\Desktop\gitmerge> git add .
PS C:\Users\manoh\Desktop\gitmerge> git commit -m "x3"
[feature 34a260e] x3
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 gamma
PS C:\Users\manoh\Desktop\gitmerge> git add .
PS C:\Users\manoh\Desktop\gitmerge> git commit -m "x4"
[feature f9ef6c2] x4
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 pie
PS C:\Users\manoh\Desktop\gitmerge> git add .
PS C:\Users\manoh\Desktop\gitmerge> git commit -m "x5"
[feature 8fb3b48] x5
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 theta
PS C:\Users\manoh\Desktop\gitmerge> git log --oneline
8fb3b48 (HEAD -> feature) x5
f9ef6c2 x4
34a260e x3
cfdba8f x2
5de7338 x1
cd53e11 c2
4d75ba7 c1
PS C:\Users\manoh\Desktop\gitmerge>

```

git add .

git commit -m "x1"

[feature 5de7338] x1

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 alpha

git add .

git commit -m "x2"

[feature cfdb8f] x2

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 beta

git add .

*PROJECT MANAGEMENT WITH GIT MANUAL (BCS358C) 3<sup>rd</sup> SEMESTER CSE 2023-24*

git commit -m "x3"

[feature 34a260e] x3

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 gamma

git add .

git commit -m "x4"

[feature f9ef6c2] x4

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 pie

git add .

git commit -m "x5"

[feature 8fb3b48] x5

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 theta

git log --oneline

8fb3b48 (HEAD -> feature) x5

f9ef6c2 x4

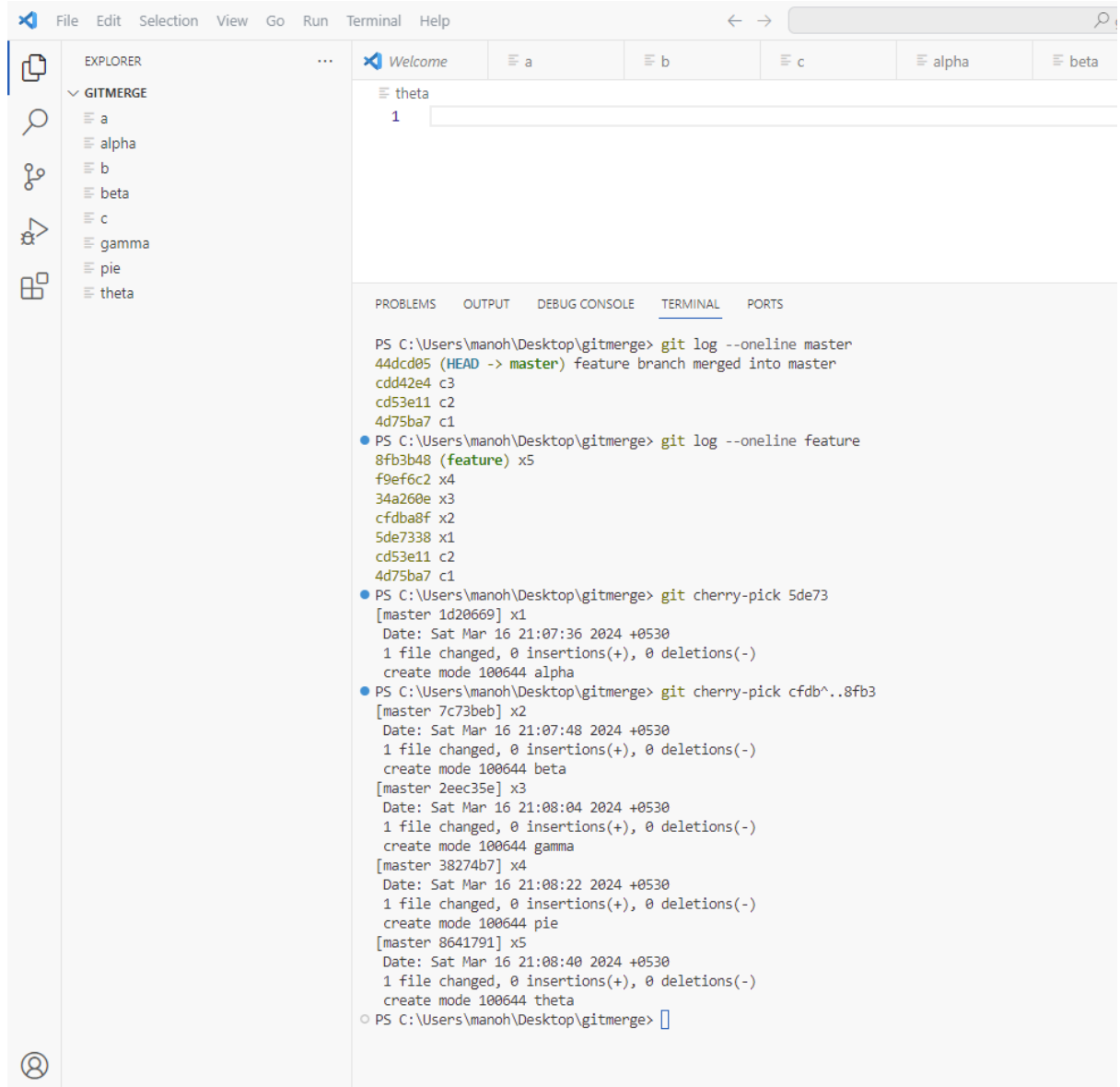
34a260e x3

cfdba8f x2

5de7338 x1

cd53e11 c2

4d75ba7 c1



git log --online master

44dcd05 (HEAD -> master) feature branch merged into master

cdd42e4 c3

cd53e11 c2

4d75ba7 c1

git log --online feature

8fb3b48 (feature) x5

f9ef6c2 x4

34a260e x3

cfdba8f x2

5de7338 x1

cd53e11 c2

4d75ba7 c1

git cherry-pick 5de73



[master 1d20669] x1

Date: Sat Mar 16 21:07:36 2024 +0530

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 alpha

git cherry-pick cfdb^..8fb3

[master 7c73beb] x2

Date: Sat Mar 16 21:07:48 2024 +0530

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 beta

[master 2eec35e] x3

Date: Sat Mar 16 21:08:04 2024 +0530

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 gamma

[master 38274b7] x4

Date: Sat Mar 16 21:08:22 2024 +0530

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 pie

[master 8641791] x5

Date: Sat Mar 16 21:08:40 2024 +0530

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 theta

**Q9. Analyzing 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?**

To view the details of a specific commit, including the author, date, and commit. message, you can use the following Git command:

git show <commit-id>

Replace <commit-id> with the actual commit hash or commit reference of the commit you want to inspect.

For example:

git show abc123

This command will display detailed information about the specified commit, including the author, date, commit message, and the changes introduced by that commit. If you only want a more concise summary of the commit information (without the changes), you can use:

git log -n 1 --pretty=format:"%h - %an, %ar : %s" <commit-id>

This command displays a one-line summary of the commit, showing the abbreviated commit hash (%h), author name (%an), relative author date (%ar), and commit message (%s).

Remember to replace <commit-id> with the actual commit hash or reference you want to inspect.

**Q10. Analyzing and Changing Git History Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."**

To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31," you can use the following git log command with the --author and since/--until options:

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

This command will display the commit history that meets the specified criteria. Adjust the author name and date range according to your requirements. The since and --until options accept a variety of date and time formats, providing flexibility in specifying the date range.

**Q11. Analyzing and Changing Git History Write the command to display the last five commits in the repository's history.**

To display the last five commits in the repository's history, you can use the following git log command with the -n option:

```
git log -n 5
```

This command shows the latest five commits in the repository, with the most recent commit displayed at the top. Adjust the number after the -n option if you want to see a different number of commits.

If you want a more concise output, you can use the oneline option:

```
git log -n 5 --oneline
```

This provides a one-line summary for each commit, including the abbreviated commit hash and the commit message.

**Q12. Analyzing and Changing Git History Write the command to undo the changes introduced by the commit with the ID "abc123".**

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

```
git revert abc123
```

Replace "abc123" with the actual commit hash or commit reference of the commit you want to undo. After running this command, Git will open a text editor for you to provide a commit message for the new revert commit.

Alternatively, if you want to completely remove a commit and all of its changes from the commit history, you can use the git reset command. However, keep in mind that using git reset can rewrite history and should be used with caution, especially if the commit has been pushed to a remote repository.

```
git reset --hard abc123
```

Again, replace "abc123" with the actual commit hash or commit reference. After using git reset --hard, your working directory will be modified to match the specified commit, discarding all commits made after it. Be cautious when using hard as it is a forceful operation and can lead to data loss.