Project Management with Git BCS358C

Q1. Setting Up and Basic Commands

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

1. Initialize a new Git repository:

```
mkdir your_project_directorycd
your_project_directory
git init
```

2. Create a new file:

```
touch your_new_file.txt
```

3. Add the file to the staging area:

```
git add your_new_file.txt
```

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 - adding a new file"
```

Q2. Creating and Managing Branches

Create a new branch named “feature-branch.” Switch to the “master” branch. Mergethe “feature-branch” into “master.”

1. 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
```

2. Switch to the “master” branch:

```
git checkout master
```

3. 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 youused git checkout -b feature branchto 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
```

4. Resolve any conflicts (if needed) and commit the merge: If there are conflicts, Git will mark the conflicted files. Open each conflicted file, resolve the conflicts, and then:

```
git add
git commit -m "Merge feature-branch into master"
```

Now, the changes from “feature-branch” are merged into the “master” branch. If youno 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 stash save "Your stash message"
```

This command will save your local changes in a temporary area, allowing you toswitch 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 stash@{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 stash@{1}
```

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/repo.git
```

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.

For SSH:

```
git clone git@github.com:example/repo.git
```

After running the git clonecommand, 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 ontothe updated remote branch.

1. Fetch the latest changes from the remote repository:

```
git fetch
```

This command fetches the latest changes from the remote repository withoutautomatically merging them into your local branches.

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

ssuming 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 modifycommits during the rebase:

```
git rebase -i origin/your-branch
```

This opens an editor where you can pick, squash, or edit individual commits.

3. Continue the rebase or resolve conflicts:

If conflicts occurred during the rebase, Git will prompt you to resolve them. After resolving conflicts, you can continue the rebase with:

```
git rebase --continue
```

If you decide to abort the rebase at any point, you can use:

```
git rebase --abort
```

4. Push the rebased branch to the remote repository:

After successfully rebasing your local branch, you may need to force-push the changes to the remote repository:

```
git push origin your-branch --force
```

Be cautious with force-pushing, especially if others are working with the same branch, as it rewrites the commit history. Now, your local branch is rebased onto the updated remote branch. Keep in mind that force-pushing should be done with caution, especially on shared branches, to avoid disrupting collaborative work.

Q6. Collaboration and Remote Repositories
Write the command to merge “feature-branch” into “master” while providing a customcommit 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. After resolving conflicts, you can continue the merge process with:

```
git merge --continue
```

Alternatively, you can use an interactive merge to modify the commit message before finalizing the merge:

```
git merge feature-branch --no-ff -e
```

This opens an editor where you can edit the commit message before completing the merge. Again, replace “feature-branch” with the name of your actual feature branch.

Q7. Git Tags and Releases
Write the command to create a lightweight Git tag named “v1.0” for a commit in yourlocal 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.

Q8. Advanced Git Operations
Write the command to cherry-pick a range of commits from “source-branch” to thecurrent branch.

To cherry-pick a range of commits from “source-branch” to the current branch, youcan use the following command:

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

Replace <start-commit>and <end-commit>with the commit hashes or referencesthat 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 yourcurrent 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.

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, includingthe 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 logcommand with the --authorand --since / --untiloptions:

```
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 --untiloptions accept a variety of date and time formats, providing flexibility in specifying the date range.

```
git log --author="Soham" --since="2024-03-07 00:00:00" --until="2024-03-08 23:59:59"
```

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 logcommand with the -noption:

```
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 -noption if you want to see a different number of commits.

If you want a more concise output, you can use the --onelineoption:

```
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 revertcommand. The git revertcommand 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 resetcommand. However, keep in mind that using git resetcan 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 --hardas it is a forceful operation and can lead to data loss.