1. Setting Up and Basic Commands

Step to initialize a new Git repository, create a new file, add it to the staging area, and commit the changes.

Open the terminal or command prompt and navigate to the directory where you want to initialize the Git repository.

To initialize a new Git repository, use the following command: **git init** This will create a new, empty Git repository in the current directory.

Now, let's create a new file in the repository. we can use any text editor you prefer. For example, if you want to create a file named "example.txt", you can use the command: **touch example.txt** This command creates an empty file named "example.txt" in the current directory. or **vi exm.txt**

After creating the file, we need to add it to the staging area. The staging area is where we specify which changes we want to include in the next commit. Use the following command to add the file to the staging area: git add example.txt This command **git add "example.txt"** to the staging area.

Finally, we can commit the changes with an appropriate commit message. A commit is like a snapshot of the current state of the repository. Use the following command to commit the changes: **git commit -m "Add example.txt file"** Replace **"Add example.txt file"** with a meaningful commit message that explains the purpose of the commit.

GIT COMMANDS:

- **git --version**

This command to check the version

- **mkdir foldername** eg: **mkdir demo**
- **cd desktop**
- **cd demo**
- **git init**
- **git status**
- **git add demo.txt**
- **git commit -m “ commiting text file”**
- **git config user.name “ ”**

- **git config user.email “ ”**
- **git config --global user.name “ ”** //To link with git hub account ,git hub username & password
- **git config – global user.email “ ”**

  Usage: git config --global user.name “ name”

  Usage: git config --global user.email “ email address”

  This command sets the author name and email address respectively to be used with your commits.

```
student@student1:~$ git --version
git version 2.34.1
student@student1:~$ mkdir 1sv22is039
student@student1:~$ cd 1sv22is039
student@student1:~/1sv22is039$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/student/1sv22is039/.git/
student@student1:~/1sv22is039$ vi demo.txt
student@student1:~/1sv22is039$ git add demo.txt
student@student1:~/1sv22is039$ git commit -m "done"
[master (root-commit) 588b566] done
 1 file changed, 1 insertion(+)
 create mode 100644 demo.txt
student@student1:~/1sv22is039$ git status
On branch master
nothing to commit, working tree clean
student@student1:~/1sv22is039$ git config --global user.name "shamanthpatil"
student@student1:~/1sv22is039$ git config --global user.email "1sv22is039@shrideviengineering.org"
student@student1:~/1sv22is039$
```

2. Creating and Managing Branches

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

Commands:

- Create a new branch named “ feature-branch” :

git branch feature-branch //This will create new branch called feature-branch

git checkout feature-branch

This command creates a new branch named "feature-branch" and switches to it.

git checkout master

- Switch back to the "master" branch:

This command switches back to the "master" branch.

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

git merge feature-branch

This command will merge the changes from "feature-branch" into the "master" branch.

After completing these steps, your "feature-branch" changes will be integrated into the "master" branch.

git status //it will check the status

git push origin master

```
student@student1:~/1sv22is039$ git branch feature-branch
student@student1:~/1sv22is039$ vi dell.txt
student@student1:~/1sv22is039$ git add dell.txt
student@student1:~/1sv22is039$ git commit -m "done"
[master 31f5779] done
 1 file changed, 1 insertion(+)
 create mode 100644 dell.txt
student@student1:~/1sv22is039$ git checkout feature-branch
Switched to branch 'feature-branch'
student@student1:~/1sv22is039$ git checkout master
Switched to branch 'master'
student@student1:~/1sv22is039$ git push origin master
Username for 'https://github.com': shamanthpatil
Password for 'https://shamanthpatil@github.com':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 466 bytes | 466.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/shamanthpatil/Sample/pull/new/master
remote:
To https://github.com/shamanthpatil/Sample.git
 * [new branch]      master -> master
```

3. Creating and Managing Branches

Commands to stash your changes, switch branches, and then apply the stashed changes

Stash your changes: **git stash** This command will temporarily save your changes, allowing you to switch branches without committing them. Git will revert your working directory and staging area to the last commit, giving you a clean state to switch branches.
Switch branches: **git checkout <branch-name> Replace <branch-name>** with the name of the branch you want to switch to. This command allows you to move to a different branch in your Git repository.
Apply the stashed changes**: git stash apply** This command reapplies the most recent stash you created. It will restore your previously stashed changes, allowing you to continue working on them.
If you have multiple stashes, you can apply a specific stash by using its index. First, list the stashes using the command **git stash list**. Then, apply a specific stash by index using the command: git **stash apply stash@{<index>} Replace <index>** with the index number of the stash you want to apply.
Additionally, if you want to remove the stash after applying it, you can use the command

git stash drop followed by the stash's index or **git stash drop stash@{<index>}.**
It's worth noting that stashing is useful when you want to switch branches without committing your changes. It allows you to work on multiple branches while keeping your changes separate and easily applicable when needed.

Commands:

git branch feature-branch2

git status

vi sample.txt

git add sample.txt

git checkout feature-branch2

git log – oneline

git stash

git stash list

git stash show stash@{1}

git stash apply

git stash pop

git stash list

```
student@student1:~/1sv22is039$ vi sham.txt
student@student1:~/1sv22is039$ git add sham.txt
student@student1:~/1sv22is039$ git commit -m "done2"
[master ca70b24] done2
 1 file changed, 1 insertion(+)
 create mode 100644 sham.txt
student@student1:~/1sv22is039$ git stash
No local changes to save
student@student1:~/1sv22is039$ git checkout feature-branch
Switched to branch 'feature-branch'
student@student1:~/1sv22is039$ git log -oneline
fatal: unrecognized argument: -oneline
student@student1:~/1sv22is039$ git log --oneline
67b960e (HEAD -> feature-branch) done
```

4. Collaboration and Remote Repositories

Clone a remote Git repository to your local machine

The process of cloning a remote Git repository to your local machine:
Open the terminal or command prompt on your local machine and navigate to the directory where you want to clone the remote repository.
Obtain the URL of the remote Git repository you want to clone. This can usually be found on the repository's webpage or by asking the repository owner. The URL will typically end with **.git**.
To clone the remote repository, use the following command: **git clone <repository-url> Replace <repository-url>** with the URL of the remote repository you obtained in the previous step. This command will create a copy of the remote repository on your local machine.
Git will start downloading the remote repository's contents to your local machine. Once the cloning process is complete, you will have a local copy of the entire repository, including its

commit history and branches.
You have successfully cloned a remote Git repository to your local machine. The cloned repository will be stored in a new directory with the same name as the remote repository.
Now, you can start working with the cloned repository on your local machine. You can make changes, create branches, commit your work, and push your changes back to the remote repository when you're ready to share or collaborate with others.
Cloning a remote repository is an essential step in collaborating on Git projects. It allows you to have a local copy of the project, work independently on your machine, and easily synchronize your changes with the remote repository.

command: git clone <repository-url>

```
student@student1:~/1sv22is039$ git clone https://github.com/shamanthpatil/shamanthpatil.github.io.git
Cloning into 'shamanthpatil.github.io'...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 32 (delta 0), reused 32 (delta 0), pack-reused 0
Receiving objects: 100% (32/32), 845.92 KiB | 4.92 MiB/s, done.
```

5. Collaboration and Remote Repositories

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

Step1:Ensure that you are in the working directory of your local repository. Run git fetch origin to fetch the latest changes from the remote repository. Replace origin with the name of your remote repository if it is different. Check out the branch you want to rebase onto the updated remote branch. For example, if you want to rebase your feature-branch onto the updated master branch, run git checkout feature-branch .

Step2:Run git **rebase** origin/master , where origin/master is the remote branch you want to rebase onto. This command replays your commits on top of the updated remote branch. If there are any conflicts during the rebase process, Git will pause the rebase and display the conflicting files. You need to resolve the conflicts manually by editing the conflicting files. After resolving the conflicts, use git add <file> for each resolved file to stage them for the rebase. Once all conflicts have been resolved and files have been staged, continue the rebase process by running git rebase – continue . This will apply the next commit on top of the updated remote

branch. If there are any further conflicts, repeat steps 5-7 until the rebase is successfully completed. After the rebase is complete, you may need to force push your updated branch to the remote repository if you have already pushed the previous commits. Use git push -f origin feature-branch to force push the updated branch. Be cautious with this step, as it rewrites the history and can cause issues for other collaborators. The local branch is now rebased onto the updated remote branch.

Commands step-by-step:

git fetch origin

git checkout feature-branch

git rebase master/origin feature-branch(Resolve any conflicts if prompted)

git add <resolved-file> (Stage each resolved file) git rebase – continue (Repeat steps 4-5 if necessary) **git push -f origin feature-branch** (Force push the updated branch to the remote repository)

Note: Replace origin with the name of your remote repository if it is different, and change featurebranch and master with the actual branch names you are working with. Be cautious with the force push (git push -f), as it rewrites history and can cause issues for other collaborators.

```
student@student1:~/1sv22is039$ git fetch origin
remote: Enumerating objects: 29, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 29 (delta 0), reused 29 (delta 0), pack-reused 0
Unpacking objects: 100% (29/29), 845.72 KiB | 1006.00 KiB/s, done.
From https://github.com/shamanthpatil/shamanthpatil.github.io
 * [new branch]      main       -> origin/main
 * [new branch]      master     -> origin/master
student@student1:~/1sv22is039$ git branch branch_x
student@student1:~/1sv22is039$ git checkout branch_x
Switched to branch 'branch_x'
student@student1:~/1sv22is039$ vi sample.txt
student@student1:~/1sv22is039$ git add sample.txt
student@student1:~/1sv22is039$ git commit -m "done"
[branch_x 92fdd3a] done
 1 file changed, 1 insertion(+)
 create mode 100644 sample.txt
student@student1:~/1sv22is039$ git rebase master branch_x
Current branch branch_x is up to date.
student@student1:~/1sv22is039$ git push -f master branch_x
fatal: 'master' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

6. Collaboration and Remote Repositories

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

First, ensure that you are currently on the "master" branch. You can switch to the "master" branch using the following command: **git checkout master**
Next, initiate the merge of the "feature-branch" into "master" by using the following command: **git merge feature-branch** This command will merge the changes from the "feature-branch" into

the "master" branch.
At this point, Git will try to automatically merge the changes. If there are any conflicts (i.e., conflicting changes in the same files), Git will notify you and prompt you to resolve the conflicts manually. You can use a text editor or a specialized merge tool to address the conflicts.
Once you have resolved the conflicts, you can proceed with the merge. Git will create a new commit to represent the merge. To provide a custom commit message for the merge, use the following command**: git commit -m "Custom commit message for merging feature-branch into master" Replace "Custom commit message for merging feature-branch into master"** with your desired commit message. Make sure to provide a meaningful message that accurately describes the purpose of the merge.
After entering the command, Git will create the merge commit with the provided custom commit message. This commit represents the merge of the changes from the "feature-branch" into the "master" branch.
You have successfully merged the "feature-branch" into the "master" branch while providing a custom commit message for the merge.
Merging branches is a crucial step in collaboration, as it brings together different sets of changes from various branches into a single branch, such as "master." It allows different team members to work on separate features or bug fixes and later integrate them into the main branch.

Commands:

git commit – m “ custom commit message”

```
student@student1:~/1sv22is039$ git merge master
Already up to date.
student@student1:~/1sv22is039$ vi dem.txt
student@student1:~/1sv22is039$ git add dem.txt
student@student1:~/1sv22is039$ git merge master
Already up to date.
student@student1:~/1sv22is039$ git commit -m "custom commit"
[branch_x 0c08f23] custom commit
 1 file changed, 1 insertion(+)
 create mode 100644 dem.txt
```

7. Git Tags and Releases

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

First, ensure that you are in the desired Git repository directory on your local machine.
Identify the commit that you want to tag. You can find the commit hash by using the **git log** command, which displays the commit history.
To create a lightweight Git tag named "v1.0" for the identified commit, use the following command**: git tag v1.0 <commit-hash> Replace <commit-hash>** with the specific commit hash that you want to tag. For example, if the commit hash is abc123, the command would be: **git tag v1.0 abc123** This command creates a lightweight tag with the name "v1.0" for the specified commit.
You have successfully created a lightweight Git tag named "v1.0" for the specific commit in your local repository.
Git tags are useful for marking specific points in history, such as releases or important milestones. Lightweight tags are simply pointers to specific commits, containing no additional metadata.
It's important to note that lightweight tags are local to your repository and are not automatically pushed to a remote repository. If you want to share the tags with others, you will need to explicitly push them to the remote repository using the git push command.
To push the "v1.0" tag to a remote repository, you can use the following command: **git push origin v1.0** Replace origin with the name of the remote repository you want to push the tag to.

Commands:

git checkout master

git tag v1.0

git tag //it will show v1.0

git tag – a v1.1 -m “ tag for release ver 1.1”

git show v1.0

git tag – l “ v1.*”

git push origin v1.0 //push tags to origin

```
student@student1:~/1sv22is039$ git tag v1.0
student@student1:~/1sv22is039$ git tag
v1.0
student@student1:~/1sv22is039$ git tag -a v1.1 -m "tag for release ver1.1"
student@student1:~/1sv22is039$ git show v1.0
commit 0330e610c54c6af20d607c0ac585585174d79a93 (HEAD -> branch_x, tag: v1.1, tag: v1.0)
Author: shamanthpatil <1sv22is039@shrideviengineering.org>
Date:   Tue Feb 27 12:52:35 2024 +0530

    done

diff --git a/hp.txt b/hp.txt
new file mode 100644
index 0000000..512e6a8
--- /dev/null
+++ b/hp.txt
@@ -0,0 +1 @@
+HEY
student@student1:~/1sv22is039$ git tag -l "v1.*"
v1.0
v1.1
student@student1:~/1sv22is039$ git push origin v1.0
Username for 'https://github.com': shamanthpatil
Password for 'https://shamanthpatil@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 303 bytes | 303.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/shamanthpatil/Sample.git
 * [new tag]         v1.0 -> v1.0
```

8. Advanced Git Operations

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

First, ensure that you are currently on the branch where you want to apply the cherry-picked commits.
Identify the range of commits you want to cherry-pick from the "source-branch". You can obtain the commit hashes or commit range using the git log command or other Git history visualization tools.
To cherry-pick a range of commits from "source-branch" to the current branch, use the following command: git **cherry-pick <start-commit>..<end-commit>** Replace <start-commit> with the hash of the first commit in the range, and <end-commit> with the hash of the last commit in the range. For example, if you want to cherry-pick the commits with hashes abc123 to def456, the command would be: **git cherry-pick abc123..def456** This command applies the changes introduced by the specified range of commits to the current branch, effectively cherry-picking them.

Commands:

Create folder Git cherry picking

cd Git cherry picking

git init

vi alpha.txt

git add . | git commit – m “ 1st commit”

vi beta.txt

git add . | git commit – m “ 2st commit”

vi gamma.txt

git add . | git commit – m “ 3st commit”

git reflog //It shows all commit history

git add .

git status

git commit – m “ all deleted”

git reflog

git cherry-pick id //----add commit history id eg:34946d4------

git log // it show all commit history with commit id

```
student@student1:~/1sv22is039$ vi alpha.txt
student@student1:~/1sv22is039$ git add alpha.txt
student@student1:~/1sv22is039$ vi beta.txt
student@student1:~/1sv22is039$ git add beta.txt
student@student1:~/1sv22is039$ vi gamma.txt
student@student1:~/1sv22is039$ git add gamma.txt
student@student1:~/1sv22is039$ git commit -m "done"
[branch_x 1bf1bae] done
 3 files changed, 3 insertions(+)
 create mode 100644 alpha.txt
 create mode 100644 beta.txt
 create mode 100644 gamma.txt
student@student1:~/1sv22is039$ git reflog
1bf1bae (HEAD -> branch_x) HEAD@{0}: commit: done
0330e61 (tag: v1.1, tag: v1.0) HEAD@{1}: rebase: checkout branch_x
0330e61 (tag: v1.1, tag: v1.0) HEAD@{2}: rebase: checkout branch_x
0330e61 (tag: v1.1, tag: v1.0) HEAD@{3}: commit: done
31f5779 (origin/master, master) HEAD@{4}: checkout: moving from master to branch_x
31f5779 (origin/master, master) HEAD@{5}: checkout: moving from feature-branch to master
e7b1f0e (feature-branch) HEAD@{6}: checkout: moving from master to feature-branch
31f5779 (origin/master, master) HEAD@{7}: commit: done
e7b1f0e (feature-branch) HEAD@{8}: commit (initial): done
student@student1:~/1sv22is039$ vi sq.txt
student@student1:~/1sv22is039$ git add sq.txt
student@student1:~/1sv22is039$ git commit -m "all deleted"
[branch_x 012a4a8] all deleted
 1 file changed, 1 insertion(+)
 create mode 100644 sq.txt
student@student1:~/1sv22is039$ git reflog
012a4a8 (HEAD -> branch_x) HEAD@{0}: commit: all deleted
1bf1bae HEAD@{1}: commit: done
0330e61 (tag: v1.1, tag: v1.0) HEAD@{2}: rebase: checkout branch_x
0330e61 (tag: v1.1, tag: v1.0) HEAD@{3}: rebase: checkout branch_x
0330e61 (tag: v1.1, tag: v1.0) HEAD@{4}: commit: done
31f5779 (origin/master, master) HEAD@{5}: checkout: moving from master to branch_x
31f5779 (origin/master, master) HEAD@{6}: checkout: moving from feature-branch to master
e7b1f0e (feature-branch) HEAD@{7}: checkout: moving from master to feature-branch
```

```
31f5779 (origin/master, master) HEAD@{6}: checkout: moving from feature-branch to master
e7b1f0e (feature-branch) HEAD@{7}: checkout: moving from master to feature-branch
31f5779 (origin/master, master) HEAD@{8}: commit: done
e7b1f0e (feature-branch) HEAD@{9}: commit (initial): done
student@student1:~/1sv22is039$ git cherry-pick 012a4a8
On branch branch_x
You are currently cherry-picking commit 012a4a8.
  (all conflicts fixed: run "git cherry-pick --continue")
  (use "git cherry-pick --skip" to skip this patch)
  (use "git cherry-pick --abort" to cancel the cherry-pick operation)

nothing to commit, working tree clean
The previous cherry-pick is now empty, possibly due to conflict resolution.
If you wish to commit it anyway, use:

    git commit --allow-empty

Otherwise, please use 'git cherry-pick --skip'
```

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?

To view the details of a specific commit in Git, including the author, date, and commit message, you can use the following command: **git show <commit ID>** Replace `<commit ID>` with the actual commit ID you want to view. This command will display the commit details, including the author's name, email, date, and the commit message.

To view the details of a specific commit in Git, you can use the `**git show**` command followed by the commit ID. Here's a step-by-step explanation of how to use Git to view the details of a commit: 1. Open your terminal or command prompt and navigate to the Git repository where the commit is located. 2. Obtain the commit ID of the specific commit you want to view. You can find the commit ID by using commands like `**git log**` or `**git reflog**`. The commit ID is a

unique identifier for each commit in Git. 3. Once you have the commit ID, use the following command to view the details of that commit: **git show <commit ID>** Replace `<commit ID>` with the actual commit ID you want to view. 4. After executing the command, Git will display the details of the commit. This includes information such as the author's name, email, date of the commit, and the commit message. The commit message is a brief description of the changes made in that commit. It provides context and helps understand the purpose of the commit. By using the `**git show**` command with the appropriate commit ID, you can easily view the details of a specific commit in Git, including the author, date, and commit message.

git log // it show all commit history with commit id

git show commitId // paste commitid

```
student@student1:~/1sv22is039$ git log
commit 8f5ff11b42ba1ec8a76b5aada8c2307f7059717b (HEAD -> branch_x, tag: v1.1, tag: v1.0)
Author: shamanthpatil <1sv22is039@shrideviengineering.org>
Date:   Mon Feb 26 14:33:01 2024 +0530

    did

commit 46c7016a5e26ecc3bce49be07a5ff78162ef649c (master)
Author: shamanthpatil <1sv22is039@shrideviengineering.org>
Date:   Mon Feb 26 14:28:09 2024 +0530

    all deleted

commit b53bb92c2efccb9150052cda58612213e1f44ee8
Author: shamanthpatil <1sv22is039@shrideviengineering.org>
Date:   Mon Feb 26 14:27:04 2024 +0530

    alpha beta gamma done

commit 588b5665479cf3f77ef9f56c0f046e8387ce7763 (feature-ranch, feature-branch2, feature-branch1, feature-branch)
Author: shamanthpatil <1sv22is039@shrideviengineering.org>
Date:   Mon Feb 26 14:14:07 2024 +0530

    done
```

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."

To list all commits made by the author "shamsiya parveen" between "2023-01-01" and "2023-12-31" in Git, you can use the following command:
git log --author="shamsiya parveen" --after="2023-01-01" --before="2023-12-31"
This command will display a list of commits made by "shamsiya parveen" within the specified date range.

git log: This is the command to view the commit history in Git.
–author="shamsiya parveen": This option filters the commit history based on the specified author name, in this case, "shamsiya parveen". Only commits made by this author will be displayed.
–after="2023-01-01": This option filters the commit history to show only the commits made after the specified date, which is "2023-01-01" in this case.
–before="2023-12-31": This option filters the commit history to show only the commits made before the specified date, which is "2023-12-31" in this case.
By combining all these options together, the command **git log --author="shamsiya parveen" --after="2023-01-01" --before="2023-12-31"** will display a list of commits made by "shamsiya parveen" between the dates "2023-01-01" and "2023-12-31".

command:

git log --author="shamsiya parveen " --after="2023-01-01" --before="2023-12-31"

11. Analysing 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 command:

git log -n 5

git log: This is the command to view the commit history in Git.
-n 5: This option limits the output to the specified number of commits, in this case, 5. It tells Git to display only the most recent 5 commits.
By running **git log -n 5**, you'll get a list of the last five commits made in the repository. Each commit will be displayed with its commit message, author, date, and a unique commit hash.

Command:
git log -n 5

```
student@student1:~/1sv22is039$ git log -n 5
commit d912de78239d16cf05ce857a92e73594d58ac57c (HEAD -> master)
Author: shamanthpatil <1sv22is039@shrideviengineering.org>
Date:   Sat Feb 24 12:45:39 2024 +0530

    alpha beta gamma done

commit dbc671212714b2ea70f9c17af730d4816c851b0f
Author: shamanthpatil <1sv22is039@shridevengineering.org>
Date:   Sat Feb 24 12:42:54 2024 +0530

    done
```

12. Analysing 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 the commit with the ID "abc123" in Git, you can use the **git revert command**. This command creates a new commit that undoes the changes made in the specified commit.
The command to undo the changes introduced by the commit with the ID "abc123" is:
textCopy code
git revert abc123
When you execute this command, Git will create a new commit that undoes the changes made in the "abc123" commit. The new commit will have a message indicating the revert and a unique commit ID. The git revert command is a safe way to undo changes since it does not rewrite the existing commit history. It adds a new commit that undoes the changes, which allows you to maintain a clear and accurate history of your project.

However, it's important to note that git revert is not the same as git reset. While git revert undoes a specific commit by creating a new commit, git reset allows you to move the branch pointer to a previous commit, effectively removing any commits after that point. **git reset** is a more powerful command and should be used with caution as it can permanently delete commits and history.

Commands:

Git revert " abc123"

Git log – oneline // it shows commit history in oneline

Git reset – hard head.1

```
student@student1:~/1sv22is039$ git reflog
d912de7 (HEAD -> master) HEAD@{0}: commit: alpha beta gamma done
dbc6712 HEAD@{1}: commit (initial): done
student@student1:~/1sv22is039$ git revert "d912de7"
[master d8206a2] Revert "alpha beta gamma done"
 Date: Sat Feb 24 12:45:39 2024 +0530
 3 files changed, 3 deletions(-)
 delete mode 100644 alpha.txt
 delete mode 100644 beta.txt
 delete mode 100644 gamma.txt
student@student1:~/1sv22is039$ git log --oneline
d8206a2 (HEAD -> master) Revert "alpha beta gamma done"
d912de7 alpha beta gamma done
dbc6712 done
```