

# PROGRAM 1

**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.**

Creating a New Directory and Navigating to It.

**STEP 1:** mkdir folder\_name:

- \* Creates a new directory with the specified name.
- \* For example, mkdir my\_project creates a directory named "my\_project".

**STEP 2:** cd folder\_name:

- \* Changes the current directory to the specified directory.
- \* For example, cd my\_project changes the current directory to "my\_project". Listing Directory Contents and Checking Current Directory

**STEP 3:** ls:

- \* Lists the contents of the current directory.
- \* This shows files and subdirectories within the current directory.

**STEP 4:** pwd:

- \* Prints the current working directory path.
- \* This tells you where you are in the file system. Initializing a Git Repository

**STEP 5:** git init:

- \* Initializes a new Git repository in the current directory.
- \* This creates a .git hidden directory to store Git's metadata. Configuring Git User Information

**STEP 6:** git config --global user.name "your\_name":

- \* Sets your name globally for all Git commits.

- \* Replace "your\_name" with your actual name.

**STEP 7:** git config --global user.email "[your\\_email@example.com](#)":

- \* Sets your email address globally for all Git commits.
- \* Replace "[your\\_email@example.com](#)" with your actual email address.

## Creating and Tracking Files

**STEP 8:** touch file1.txt:

- \* Creates a new file named "file1.txt" in the current directory.

**STEP 9:** ls:

- \* Lists the contents of the current directory.
- \* This shows files and subdirectories within the current directory

**STEP 10:** git status:

- \* Shows the current state of the working directory.
- \* It indicates which files are tracked, untracked, modified, or staged.

**STEP 11:** git add file1.txt:

- \* Stages the changes in "file1.txt" for the next commit.
- \* Staged changes are included in the next commit.

**STEP 12:** git status:

- \* Shows the current state of the working directory.
- \* It indicates which files are tracked, untracked, modified, or staged.

**STEP 13:** git commit -m "file created":

- \* Creates a new commit with the specified message.
  - \* The staged changes are committed to the repository.
- Editing and Committing a File

**STEP 14:** git status:

- \* Shows the current state of the working directory.
- \* It indicates which files are tracked, untracked, modified, or staged.

**STEP 15:** vi file2.txt:

- \* Opens the file "file2.txt" in the vi text editor.
- \* You can edit the file's content.

**STEP 16:** git add .:

- \* Stages all changes in the current directory for the next commit.

**STEP 17:** git commit -m "added file":

- \* Creates a new commit with the specified message.
- \* All staged changes are committed to the repository.

## Viewing Commit History

**STEP 18:** git status:

- \* Shows the current state of the working directory.
- \* It indicates which files are tracked, untracked, modified, or staged.

**STEP 19:** git log:

- \* Shows the commit history of the current repository.

It displays information about each commit, such as the commit hash, author, date, and commit message.

These commands are fundamental to using Git for version control. By mastering these commands, you can effectively manage your projects and collaborate with others.

## OUTPUT:

```
bash CSBS x
Prajwals-MacBook-Air:git admin$ mkdir CSBS
Prajwals-MacBook-Air:git admin$ cd CSBS
Prajwals-MacBook-Air:CSBS admin$ git init
Initialized empty Git repository in /Users/admin/Documents/prajwal/Gitdemo/git/CSBS/.git/
Prajwals-MacBook-Air:CSBS admin$
Prajwals-MacBook-Air:CSBS admin$ git config --global user.name "CSBS 2023"
Prajwals-MacBook-Air:CSBS admin$ git config --global user.email "CSBSVTU2023@gmail.com"
Prajwals-MacBook-Air:CSBS admin$ touch file1.txt
Prajwals-MacBook-Air:CSBS admin$ ls
file1.txt
Prajwals-MacBook-Air:CSBS admin$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

nothing added to commit but untracked files present (use "git add" to track)
Prajwals-MacBook-Air:CSBS admin$ git add file1.txt
Prajwals-MacBook-Air:CSBS admin$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt

Prajwals-MacBook-Air:CSBS admin$ git commit -m "File created"
[main (root-commit) bfc2a6c] File created
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
● Prajwals-MacBook-Air:CSBS admin$ vi file2.txt
● Prajwals-MacBook-Air:CSBS admin$ git add .
● main 0 0 0

Prajwals-MacBook-Air:CSBS admin$ git commit -m "File created"
[main (root-commit) bfc2a6c] File created
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
● Prajwals-MacBook-Air:CSBS admin$ vi file2.txt
● Prajwals-MacBook-Air:CSBS admin$ git add .
● Prajwals-MacBook-Air:CSBS admin$ git commit -m "Added file"
On branch main
nothing to commit, working tree clean
● Prajwals-MacBook-Air:CSBS admin$ git status
On branch main
nothing to commit, working tree clean
● Prajwals-MacBook-Air:CSBS admin$ git log
commit bfc2a6c4d4aba7bb81d1037a8d74bb1b8b5e2f1e (HEAD -> main)
Author: CSBS 2023 <CSBSVTU2023@gmail.com>
Date:   Fri Dec 20 22:52:21 2024 +0530

    File created
○ Prajwals-MacBook-Air:CSBS admin$
```

## PROGRAM 2

**Create a new branch named "new" Switch to the "master" branch.**

**Merge the "new" into "master."**

**STEP 1:** git branch new:

\* This command creates a new branch named "new". This new branch is a copy of the current branch's state.

**STEP 2:** git branch:

\* This command lists all the branches in the current repository. You should see both "master" and "new" branches now.

**STEP 3:** git checkout new:

\* This command switches your working directory to the newly created "new" branch. Any changes you make from now on will be specific to this branch.

**STEP 4:** vi file3.txt

\* This command opens the file "file3.txt" in the vi text editor. You can edit the contents of this file.

**STEP 5:** git add .:

\* This command stages all the changes made to tracked files in the current directory. In this case, it stages the changes made to "file3.txt".

**STEP 6:** git commit -m "edited file":

\* This command creates a new commit on the "new" branch with the message "edited file". The staged changes are included in this commit.

**STEP 7:** git status:

\* This command shows the current state of the working directory. It will show that you're on the "new" branch and there are no changes to be committed.

**STEP 8:** git checkout master:

\* This command switches your working directory back to the "master" branch. Any changes you make from now on will be specific to this branch.

**STEP 9:** git merge new:

\* This command merges the changes from the "new" branch into the "master" branch. Git will automatically try to combine the changes from both branches. If there are conflicts, you'll need to resolve them manually.

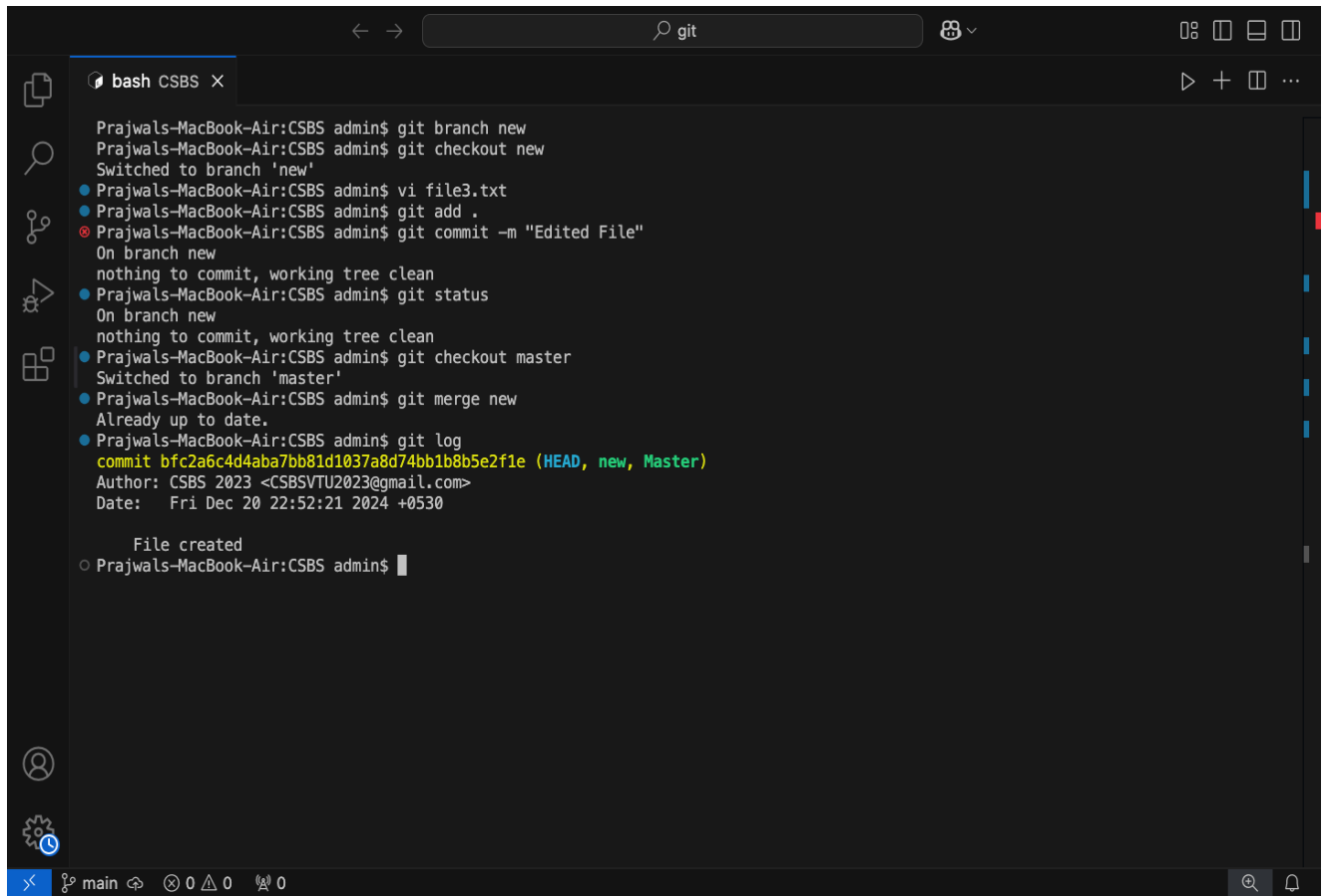
**STEP 10:** git log:

\* This command shows the commit history of the current branch. You should see the commit you made on the "new" branch, followed by the merge commit that combined the changes from "new" into "master". In essence, what we've done is:

- \* Created a new branch to work on a specific feature or bug fix.
- \* Made changes to the file "file3.txt" on the new branch.
- \* Committed the changes to the new branch.
- \* Merged the changes from the new branch back into the main branch.

This is a common workflow in Git for managing different development tasks and keeping your codebase organized.

## OUTPUT:



```
Prajwals-MacBook-Air:CSBS admin$ git branch new
Prajwals-MacBook-Air:CSBS admin$ git checkout new
Switched to branch 'new'
Prajwals-MacBook-Air:CSBS admin$ vi file3.txt
Prajwals-MacBook-Air:CSBS admin$ git add .
Prajwals-MacBook-Air:CSBS admin$ git commit -m "Edited File"
On branch new
nothing to commit, working tree clean
Prajwals-MacBook-Air:CSBS admin$ git status
On branch new
nothing to commit, working tree clean
Prajwals-MacBook-Air:CSBS admin$ git checkout master
Switched to branch 'master'
Prajwals-MacBook-Air:CSBS admin$ git merge new
Already up to date.
Prajwals-MacBook-Air:CSBS admin$ git log
commit bfc2a6c4d4aba7bb81d1037a8d74bb1b8b5e2f1e (HEAD, new, Master)
Author: CSBS 2023 <CSBSVTU2023@gmail.com>
Date:   Fri Dec 20 22:52:21 2024 +0530

    File created
Prajwals-MacBook-Air:CSBS admin$
```

## PROGRAM 3

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

Let's break down each command step-by-step:

**STEP 1:** git branch new:

- \* This command creates a new branch named "new". This new branch is a copy of the current branch's state.

**STEP 2:** git checkout new:

- \* This command switches your working directory to the newly created "new" branch. Any changes you make from now on will be specific to this branch. **STEP 3:** vi file4.txt:

- \* This command opens the file "file4.txt" in the vi text editor. You can edit the contents of this file.

**STEP 4:** git stash:

- \* This command temporarily saves your uncommitted changes. This is useful when you need to switch branches or perform other tasks without committing your current work.

**STEP 5:** git checkout master:

- \* This command switches your working directory back to the "master" branch.

**STEP 6:** git stash apply:

- \* This command applies the saved changes from the stash to the current branch, which is "master" in this case.

**STEP 7:** git status:



- \* This command shows the current state of the working directory. It will show that there are changes to be committed.

**STEP 8:** git add .:

- \* This command stages all the changes made to tracked files in the current directory, which are the changes applied from the stash.

**STEP 9:** git status:

- \* This command shows the current state of the working directory. It will show that there are staged changes to be committed.

**STEP 10:** git commit -m "stash changed":

- \* This command creates a new commit on the "master" branch with the message "stash changed". The staged changes are included in this commit.

**STEP 11:** git log:

- \* This command shows the commit history of the current branch. You should see the commit you just made, which includes the changes from the stash. In essence, what we've done is:

- \* Created a new branch to work on a specific feature or bug fix.
- \* Made changes to the file "file4.txt" on the new branch.
- \* Temporarily saved the changes using git stash.
- \* Switched back to the main branch.
- \* Applied the saved changes to the main branch using git stash apply.
- \* Committed the applied changes to the main branch.

This workflow is useful when you need to switch between tasks or branches without losing your current work.

By stashing your changes, you can temporarily save them and restore them later when needed.

## OUTPUT:

```
bash CSBS x
Prajwals-MacBook-Air:git admin$ mkdir CSBS
Prajwals-MacBook-Air:git admin$ cd CSBS
Prajwals-MacBook-Air:CSBS admin$ git branch new
Prajwals-MacBook-Air:CSBS admin$ git checkout new
D   file1.txt
Switched to branch 'new'
Prajwals-MacBook-Air:CSBS admin$ vi file3.txt
Prajwals-MacBook-Air:CSBS admin$ git stash
Saved working directory and index state WIP on new: 7c5187a File created
Prajwals-MacBook-Air:CSBS admin$ git checkout main
Switched to branch 'main'
Prajwals-MacBook-Air:CSBS admin$ git stash apply
On branch main
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    ../file1.txt

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

no changes added to commit (use "git add" and/or "git commit -a")
Prajwals-MacBook-Air:CSBS admin$ git status
On branch main
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    ../file1.txt

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

no changes added to commit (use "git add" and/or "git commit -a")
Prajwals-MacBook-Air:CSBS admin$ git add .
```

```
Prajwals-MacBook-Air:CSBS admin$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   file3.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    ../file1.txt

Prajwals-MacBook-Air:CSBS admin$
Prajwals-MacBook-Air:CSBS admin$ git commit -m "Stash Changed"
[main f24b09d] Stash Changed
 1 file changed, 1 insertion(+)
 create mode 100644 CSBS/file3.txt
Prajwals-MacBook-Air:CSBS admin$ git log
commit f24b09d821d105103f8d9007aea7e4f1a4450eee (HEAD -> main)
Author: CSBS 2023 <CSBSVTU2023@gmail.com>
Date:   Fri Dec 20 23:14:19 2024 +0530

    Stash Changed

commit 7c5187a42aba4dd0c1b8bf38beba3400544f987e (new)
Author: Prajwal <Xyz@gmail.com>
Date:   Thu Dec 19 01:41:44 2024 +0530

    File created
Prajwals-MacBook-Air:CSBS admin$
```

## PROGRAM 4

### Clone a remote Git repository to your local machine.

#### Breaking Down the Commands

Let's break down each command step-by-step:

#### **STEP 1:** mkdir clone:

- \* mkdir: This is a command-line utility used to create a new directory.
- \* clone: This is the name of the directory you want to create. So, this command creates a new directory named "clone" in your current working directory.

#### **STEP 2:** cd clone:

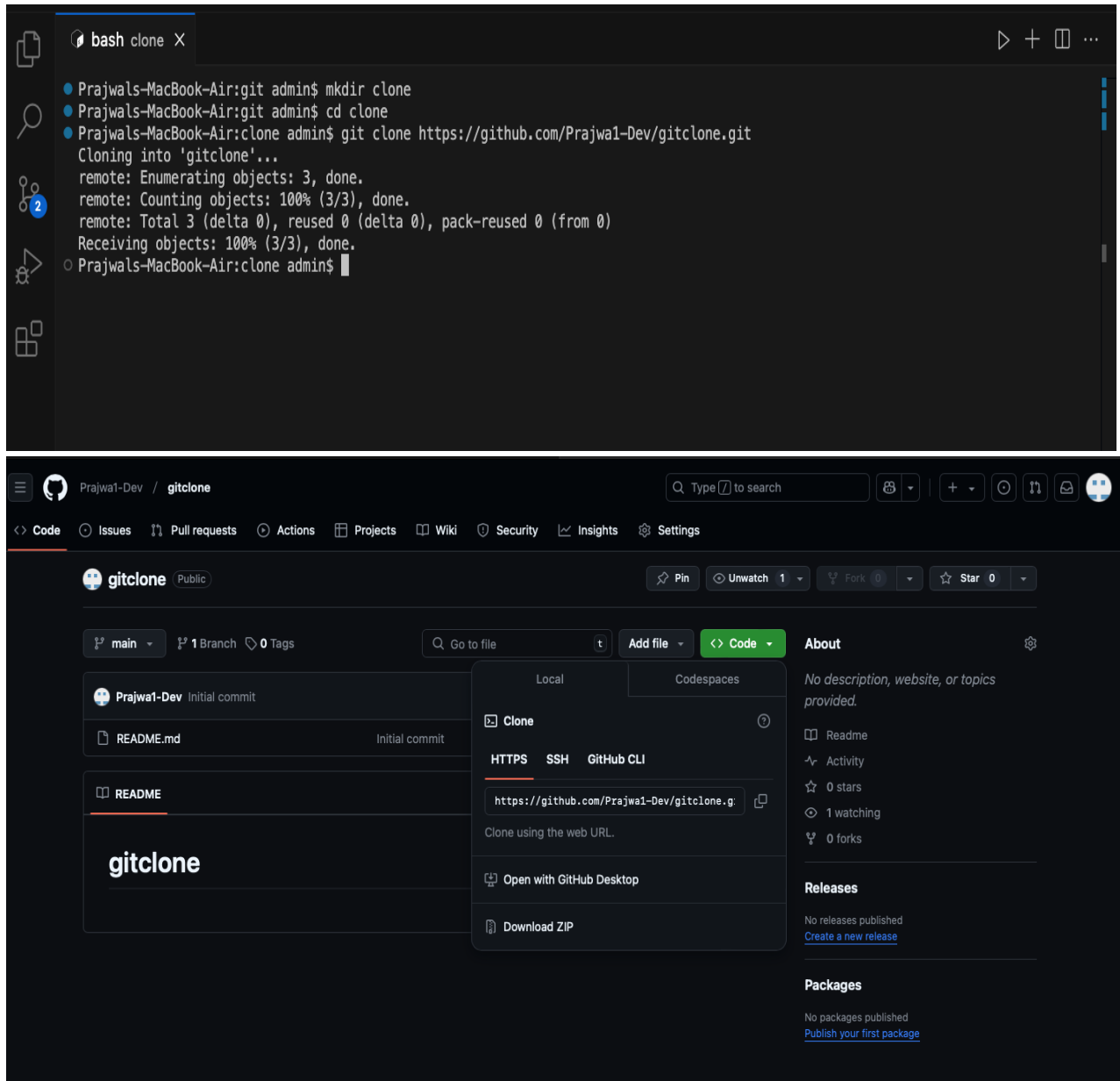
- \* cd: This command is used to change your current working directory.
  - \* clone: This is the name of the directory you want to change into.
- This command will move you into the newly created "clone" directory.

#### **STEP 3:** git clone (paste URL):

- \* Create Github credentials
- \* git clone: This Git command is used to clone an existing Git repository.
- \* (paste URL): This is where you would paste the URL of the Git repository you want to clone.

This command will fetch the entire history of the remote repository and create a local copy of it within the "clone" directory.

## OUTPUT:



The image displays two screenshots related to a GitHub repository named 'gitclone'.

The top screenshot is a terminal window titled 'bash clone'. It shows the following commands and output:

```
Prajwals-MacBook-Air:git admin$ mkdir clone
Prajwals-MacBook-Air:git admin$ cd clone
Prajwals-MacBook-Air:clone admin$ git clone https://github.com/Prajwa1-Dev/gitclone.git
Cloning into 'gitclone'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
Prajwals-MacBook-Air:clone admin$
```

The bottom screenshot is a screenshot of the GitHub repository page for 'gitclone' by 'Prajwa1-Dev'. The repository is public and has 1 branch (main) and 0 tags. The 'main' branch is selected. The repository contains a 'README.md' file, which is the 'Initial commit'. The 'README.md' file content is displayed, showing the repository name 'gitclone'. The 'About' section on the right indicates that no description, website, or topics are provided. The 'Releases' section shows no releases published, with a link to 'Create a new release'. The 'Packages' section shows no packages published, with a link to 'Publish your first package'.

## PROGRAM 5

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

Understanding the Commands

Let's break down each command step-by-step:

**STEP 1:** cd repository name:

- \* Purpose: This command changes the current directory to the specified repository.
- \* Explanation:
- \* cd: Stands for "change directory."
- \* repository name: Replace this with the actual name of your Git repository.
- \* Example: cd my-project This would change your current working directory to the folder named "my-project".

**STEP 2:** git fetch origin:

- \* Purpose: This command fetches the latest changes from the remoterepository named "origin" to your local repository.
- \* Explanation:
- \* git fetch: Fetches commits, files, and other objects from a remote repository.
- \* origin: Refers to the default remote repository, usually set up when you clone a repository.
- \* What happens:
- \* It downloads new commits, branches, and tags from the remoterepository.
- \* These changes are stored locally but are not immediately merged into your current branch.

**STEP 3:** git rebase origin/main:

- \* Purpose: This command reapplies your local commits on top of the latest commits from the remote "main" branch.
- \* Explanation:
  - \* git rebase: Rebases your current branch onto another branch.
  - \* origin/main: Refers to the remote "main" branch.
- \* What happens:
  - \* It replays your local commits on top of the latest commits from the remote "main" branch.
  - \* This creates a linear commit history, making it easier to review and understand the changes.

#### **STEP 4:** git log:

- \* Purpose: This command displays a log of commits in your local repository.
- \* Explanation:
  - \* git log: Shows the commit history.
- \* What happens:
  - \* It displays a list of commits, including their hash, author, date, and commit message.
  - \* You can use various options with git log to customize the output, such as filtering by author, date, or commit message.
- Key Points to Remember:**
  - \* Always fetch before rebasing to ensure you have the latest changes from the remote repository.
  - \* Rebasing can be a powerful tool, but it can also be complex. Use it with caution, especially in shared repositories.
  - \* Git log is a valuable tool for understanding the history of your project and identifying specific changes.

By understanding these commands and their functions, you can effectively manage your Git workflow and keep your local repository up-to-date with the remote repository.

## OUTPUT:

```
bash gitclone X
Prajwals-MacBook-Air:git admin$ git clone https://github.com/Prajwal-Dev/gitclone.git
Cloning into 'gitclone'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
Prajwals-MacBook-Air:git admin$ cd gitclone
Prajwals-MacBook-Air:gitclone admin$ git fetch origin
Prajwals-MacBook-Air:gitclone admin$ git rebase origin/main
Current branch main is up to date.
Prajwals-MacBook-Air:gitclone admin$ git log
commit 64f91b267a663a3c974babb0df8e71b3356d69d (HEAD -> main, origin/main, origin/HEAD)
Author: Prajwal <prajwalmasheewadi.99@gmail.com>
Date: Thu Dec 19 12:55:15 2024 +0530

    Initial commit
Prajwals-MacBook-Air:gitclone admin$
```

## PROGRAM 6

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

**STEP 1:** git branch feature-branch:

- \* Purpose: Creates a new branch named "feature-branch" off of the currently checked-out branch.
- \* How it works:
  - \* Git creates a new pointer to the current commit.
  - \* This new pointer is named "feature-branch."
  - \* You can now make changes on this new branch without affecting the original branch.

**STEP 2:** git checkout feature-branch:

- \* Purpose: Switches the active branch to "sub-branch."
- \* How it works:
  - \* Git updates the working directory to match the state of the "sub-branch."
  - \* Any changes you make from this point on will be applied to the "sub-branch."

**STEP 3:** vi file5.txt:

- \* Purpose: Opens the file "file5.txt" in the vi text editor.
- \* How it works:
  - \* This command is not specific to Git but is a common Linux command to edit text files.
  - \* You can use vi to make changes to the file, which will be reflected in the "sub-branch" when you commit.

**STEP 4:** git commit -m "commit":

Commits the staged changes with the message "commit".



**STEP 5:** git checkout master:

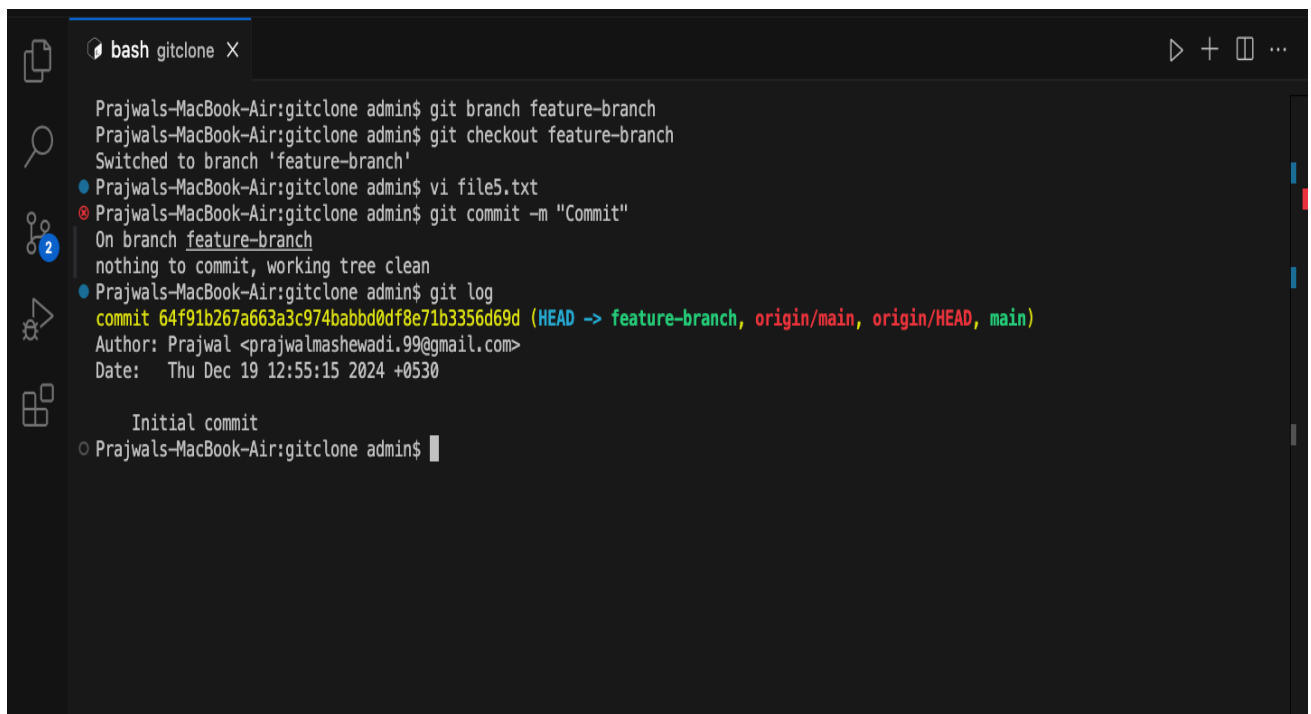
Switches to the master branch (or main, depending on your project setup).

**STEP 6:** git merge feature-branch:

Merges the changes from feature-branch into the current branch (master in this case).

**STEP 7:** git log:

- \* Purpose: Shows the commit history of the current branch.
- \* How it works:
- \* Git displays a list of commits, starting from the most recent.
- \* Each commit is shown with its hash, author, date, and a brief message.
- \* This command helps you understand the history of your project and identify specific changes.

**OUTPUT:**A terminal window titled 'bash gitclone X' with a dark background. The terminal shows a series of git commands and their outputs. The commands are: 'git branch feature-branch', 'git checkout feature-branch', 'vi file5.txt', 'git commit -m "Commit"', and 'git log'. The outputs include 'Switched to branch 'feature-branch'', 'On branch feature-branch', 'nothing to commit, working tree clean', and the details of an initial commit with a long hash, author, and date. The terminal also shows a file explorer on the left with icons for files, search, and git operations.

```
Prajwals-MacBook-Air:gitclone admin$ git branch feature-branch
Prajwals-MacBook-Air:gitclone admin$ git checkout feature-branch
Switched to branch 'feature-branch'
Prajwals-MacBook-Air:gitclone admin$ vi file5.txt
Prajwals-MacBook-Air:gitclone admin$ git commit -m "Commit"
On branch feature-branch
nothing to commit, working tree clean
Prajwals-MacBook-Air:gitclone admin$ git log
commit 64f91b267a663a3c974babb0df8e71b3356d69d (HEAD -> feature-branch, origin/main, origin/HEAD, main)
Author: Prajwal <prajwalmashewadi.99@gmail.com>
Date: Thu Dec 19 12:55:15 2024 +0530

Initial commit
Prajwals-MacBook-Air:gitclone admin$
```

## PROGRAM 7

**Write the command to create a lightweight git tag named “V1.0” for a commit in your local repository.**

**STEP 1:** `git tag V1.0 <commit_id>`:

\* This command creates a tag named "V1.0" and associates it with the specified commit ID. This tag marks a specific point in your project's history, often representing a release version.

**STEP 2:** `git show V1.0`:

\* This command displays detailed information about the commit tagged as "V1.0". It shows the commit message, author, date, and the changes introduced in that commit.

**STEP 3:** `git tag V1.1 <commit_id>`:

\* This command creates another tag named "V1.1" and associates it with a different commit ID. This might represent a subsequent release or a major update.

**STEP 4:** `git show V1.1`:

\* Similar to the previous command, this displays detailed information about the commit tagged as "V1.1".

\* **Release Management:** Tags help identify specific releases and their corresponding code.

\* **Historical Reference:** You can easily reference past versions of your project.

\* **Code Review:** Tags can be used to mark specific points for review or comparison.

\* **Rollback:** In case of issues, you can revert to a tagged version.

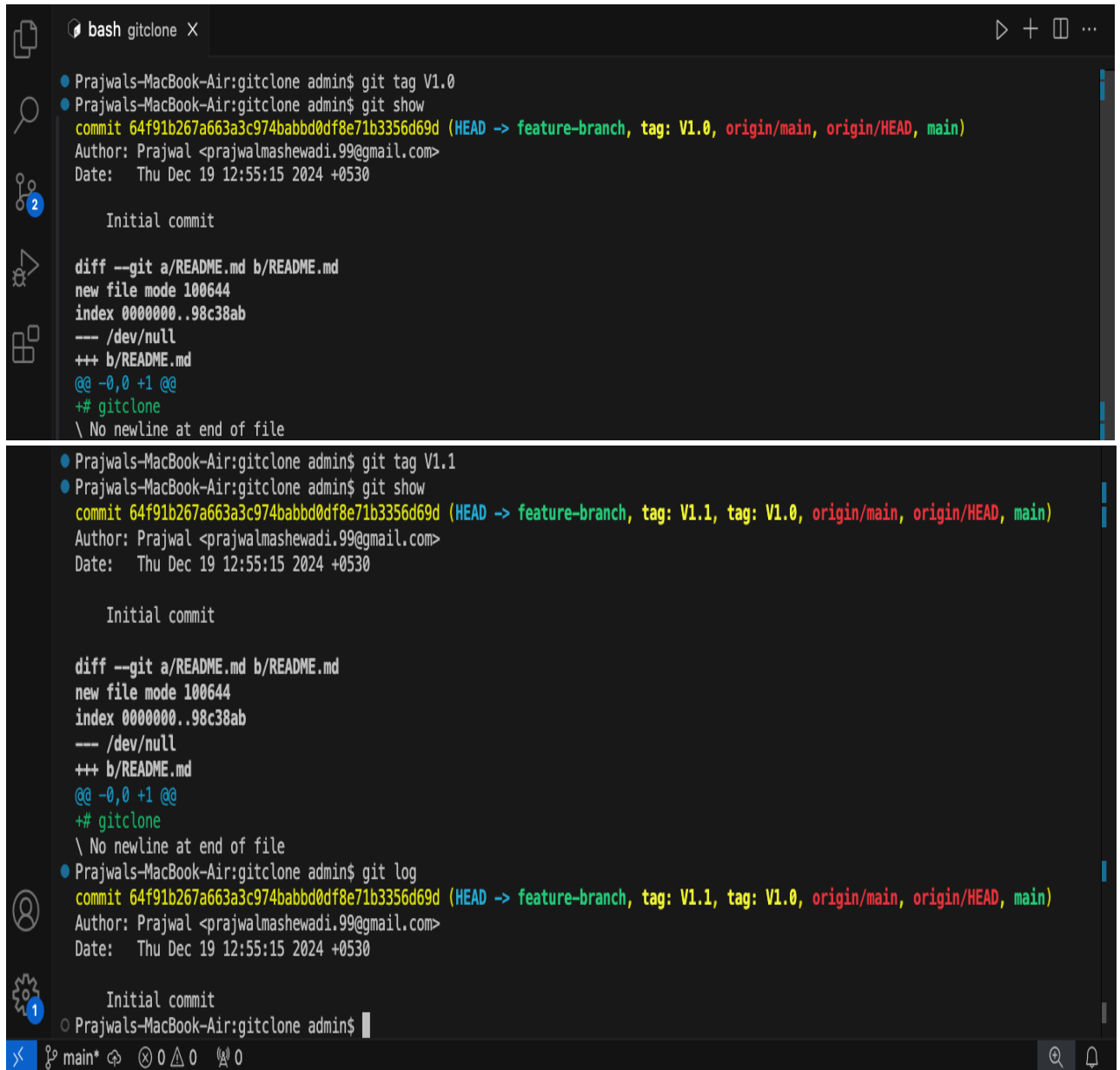
\* **Lightweight Tags:** The commands above create lightweight tags, which are simply pointers to commits.

\* **Annotated Tags:** You can create annotated tags using the `-a` flag, which allows you to add additional metadata, such as a tagger, date, and a message.

\* **Tagging the Latest Commit:** To tag the latest commit, you can use the HEAD reference:  
`git tag V1.0 HEAD`

\* Pushing Tags to a Remote Repository: `git push origin V1.0 V1.1` By effectively using Git tags, you can maintain a clear and organized project history, making it easier to manage releases, track changes, and collaborate with others.

## OUTPUT:



```
bash gitclone X
Prajwals-MacBook-Air:gitclone admin$ git tag V1.0
Prajwals-MacBook-Air:gitclone admin$ git show
commit 64f91b267a663a3c974babbd0df8e71b3356d69d (HEAD -> feature-branch, tag: V1.0, origin/main, origin/HEAD, main)
Author: Prajwal <prajwalmasheewadi.99@gmail.com>
Date: Thu Dec 19 12:55:15 2024 +0530

Initial commit

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..98c38ab
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+# gitclone
\ No newline at end of file

Prajwals-MacBook-Air:gitclone admin$ git tag V1.1
Prajwals-MacBook-Air:gitclone admin$ git show
commit 64f91b267a663a3c974babbd0df8e71b3356d69d (HEAD -> feature-branch, tag: V1.1, tag: V1.0, origin/main, origin/HEAD, main)
Author: Prajwal <prajwalmasheewadi.99@gmail.com>
Date: Thu Dec 19 12:55:15 2024 +0530

Initial commit

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..98c38ab
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+# gitclone
\ No newline at end of file

Prajwals-MacBook-Air:gitclone admin$ git log
commit 64f91b267a663a3c974babbd0df8e71b3356d69d (HEAD -> feature-branch, tag: V1.1, tag: V1.0, origin/main, origin/HEAD, main)
Author: Prajwal <prajwalmasheewadi.99@gmail.com>
Date: Thu Dec 19 12:55:15 2024 +0530

Initial commit
Prajwals-MacBook-Air:gitclone admin$
```

## PROGRAM 8

**Write the command to cherry-pick a range of commits from “main” to the current branch.**

Understanding the Commands

Let's break down each command and its purpose within a Git repository:

**STEP 1:** git checkout main:

- \* Purpose: Switches the active branch to "main".
- \* How it works:
- \* Git updates the working directory to match the state of the "main" branch.
- \* Any changes you make from this point on will be applied to the "main" branch.

**STEP 2:** git cherry-pick (paste commit ID):

- \* Purpose: Applies the changes introduced by a specific commit to the current branch.
- \* How it works:
- \* You provide the commit ID of the change you want to cherry-pick.
- \* Git identifies the changes introduced by that commit.
- \* It creates a new commit on the current branch with those changes.
- \* This is useful when you want to apply a specific fix or feature from one branch to another.

**STEP 3:** git log:

- \* Purpose: Shows the commit history of the current branch.
- \* How it works:
- \* Git displays a list of commits, starting from the most recent.
- \* Each commit is shown with its hash, author, date, and a brief message.

- \* This command helps you understand the history of your project and identify specific changes.

Example Scenario:

Imagine you have a "feature-branch" where you've made a great improvement. You want to apply this improvement to the "main" branch without merging the entire "feature-branch."

- \* Switch to the "main" branch:

git checkout main \* Cherry-pick the commit: git cherry-pick <commit-id>

Replace <commit-id> with the actual ID of the commit you want to cherry-pick. You can find the commit ID from the git log output.

- \* Verify the changes:

git log

You should now see the cherry-picked commit in the log of the "main" branch. Important Considerations:

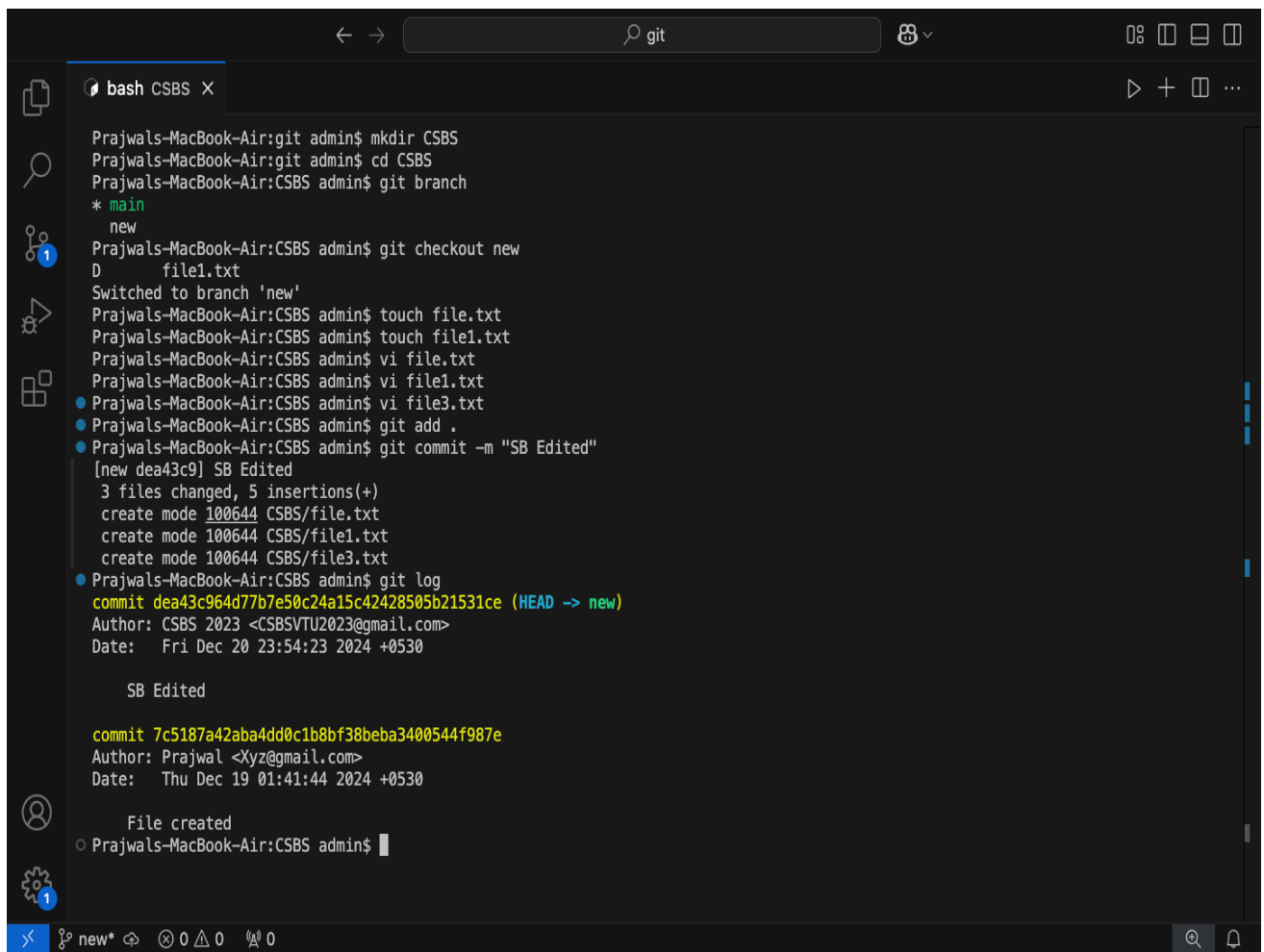
- \* Conflict Resolution: If there are conflicts between the cherry-picked commit and the current state of the "main" branch, Git will prompt you to resolve them manually.

- \* Multiple Commits: You can cherry-pick multiple commits by providing their respective IDs.

- \* Selective Cherry-Picking: Be cautious when cherry-picking commits, as it can introduce unintended side effects if the commits are tightly coupled with other changes.

By understanding these commands, you can effectively manage your Git workflow and selectively apply changes between branch.

## OUTPUT:



```
Prajwals-MacBook-Air:git admin$ mkdir CSBS
Prajwals-MacBook-Air:git admin$ cd CSBS
Prajwals-MacBook-Air:CSBS admin$ git branch
* main
new
Prajwals-MacBook-Air:CSBS admin$ git checkout new
D      file1.txt
Switched to branch 'new'
Prajwals-MacBook-Air:CSBS admin$ touch file.txt
Prajwals-MacBook-Air:CSBS admin$ touch file1.txt
Prajwals-MacBook-Air:CSBS admin$ vi file.txt
Prajwals-MacBook-Air:CSBS admin$ vi file1.txt
Prajwals-MacBook-Air:CSBS admin$ vi file3.txt
Prajwals-MacBook-Air:CSBS admin$ git add .
Prajwals-MacBook-Air:CSBS admin$ git commit -m "SB Edited"
[new dea43c9] SB Edited
3 files changed, 5 insertions(+)
create mode 100644 CSBS/file.txt
create mode 100644 CSBS/file1.txt
create mode 100644 CSBS/file3.txt
Prajwals-MacBook-Air:CSBS admin$ git log
commit dea43c964d77b7e50c24a15c42428505b21531ce (HEAD -> new)
Author: CSBS 2023 <CSBSVTU2023@gmail.com>
Date:   Fri Dec 20 23:54:23 2024 +0530

    SB Edited

commit 7c5187a42aba4dd0c1b8bf38beba3400544f987e
Author: Prajwal <Xyz@gmail.com>
Date:   Thu Dec 19 01:41:44 2024 +0530

    File created
Prajwals-MacBook-Air:CSBS admin$
```

## PROGRAM 9

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

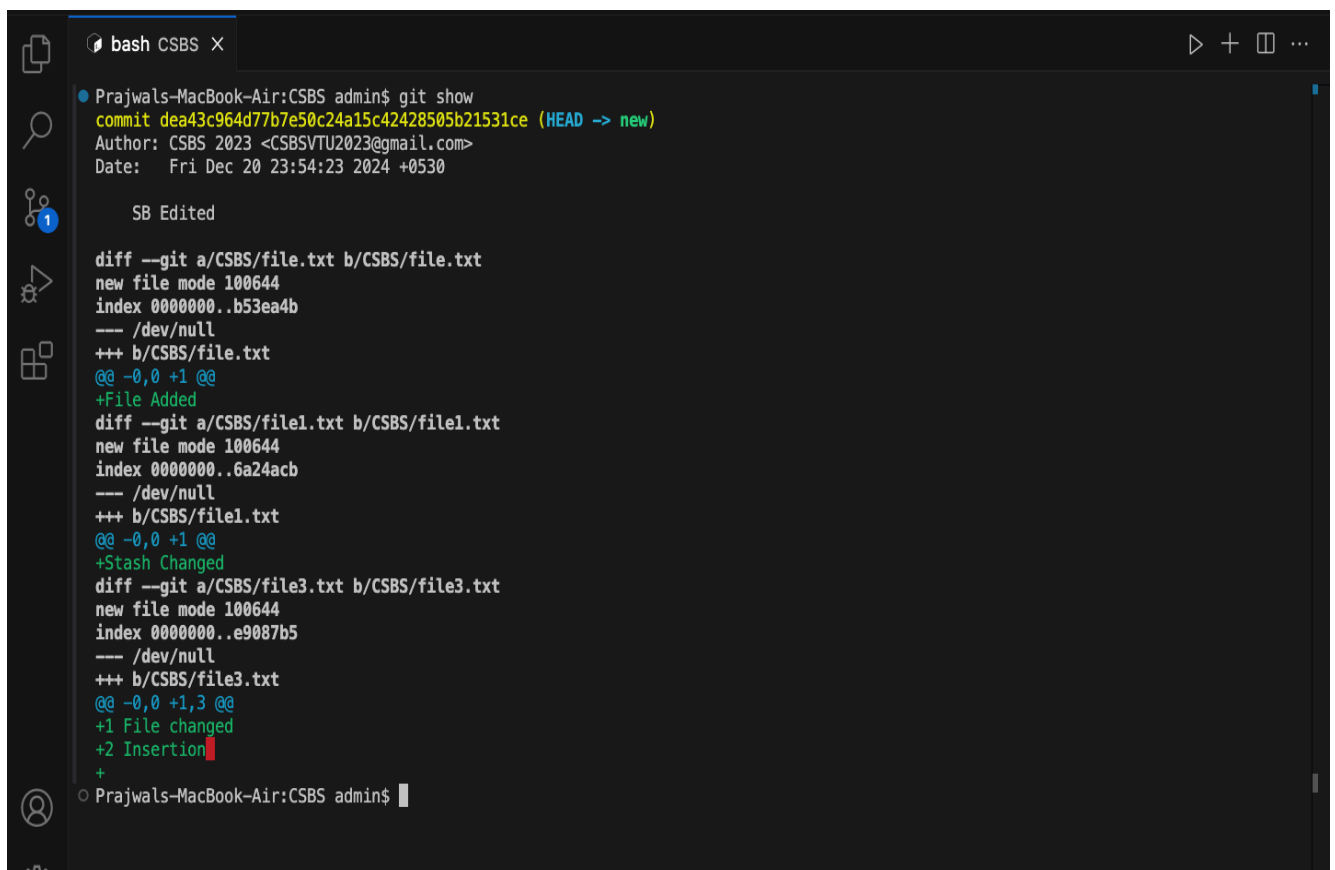
**STEP 1:** git show <commit ID>:

**git show** displays detailed information about a Git object, most commonly a commit.

By default, it shows the commit hash, author, date, commit message, and diff of the changes.

It can be used with a variety of arguments (commit hash, branch, tag, etc.) to inspect different parts of the repository's history.

### OUTPUT:



```
bash CSBS x
Prajwals-MacBook-Air:CSBS admin$ git show
commit dea43c964d77b7e50c24a15c42428505b21531ce (HEAD -> new)
Author: CSBS 2023 <CSBSVTU2023@gmail.com>
Date:   Fri Dec 20 23:54:23 2024 +0530

    SB Edited

diff --git a/CSBS/file.txt b/CSBS/file.txt
new file mode 100644
index 0000000..b53ea4b
--- /dev/null
+++ b/CSBS/file.txt
@@ -0,0 +1 @@
+File Added
diff --git a/CSBS/file1.txt b/CSBS/file1.txt
new file mode 100644
index 0000000..6a24acb
--- /dev/null
+++ b/CSBS/file1.txt
@@ -0,0 +1 @@
+Stash Changed
diff --git a/CSBS/file3.txt b/CSBS/file3.txt
new file mode 100644
index 0000000..e9087b5
--- /dev/null
+++ b/CSBS/file3.txt
@@ -0,0 +1,3 @@
+1 File changed
+2 Insertion
+
Prajwals-MacBook-Air:CSBS admin$
```

## PROGRAM 10

**Write the command to list all commits made by the author “Mokshith” on “2024-12-04”.**

**STEP 1:** `git log --author="author name" --since="start date" -until="end date":`

This Git command is a powerful tool for filtering and analyzing your project's commit history. It allows you to view specific commits based on the author, date range, and other criteria. Breakdown of the Command:

- \* `git log`: This is the basic command to view the commit history.
  - \* `--author="author name"`: This flag filters the commits to only show those authored by the specified "author name."
  - \* `--since="start date"`: This flag limits the log to commits made after the specified "start date."
  - \* `--until="end date"`: This flag limits the log to commits made before the specified "end date."
- How to Use It:
- \* Specify the Author:
    - \* Replace "author name" with the actual name of the author you want to filter by. You can use partial names or email addresses as well.
  - \* Specify the Date Range:
    - \* Replace "start date" and "end date" with the desired dates. You can use various date formats, such as:
      - \* YYYY-MM-DD (e.g., 2023-11-22)
      - \* relative dates (e.g., yesterday, 2 weeks ago)
      - \* specific time periods (e.g., 2023-01-01..2023-12-31)

Example:

To view all commits made by "John Doe" between January 1st, 2023, and December 31st, 2023: `git log --author="John Doe" --since="2023-01-01" --until="2024-01-01"`



## OUTPUT:

```
bash CSBS x
Prajwals-MacBook-Air:CSBS admin$ git log --author="CSBS" --since="2024-12-20" --until="2024-12-20"
fatal: unrecognized argument: --until-2024-12-20
Prajwals-MacBook-Air:CSBS admin$ git log --author="CSBS"
commit dea43c964d77b7e50c24a15c42428505b21531ce (HEAD -> new)
Author: CSBS 2023 <CSBSVTU2023@gmail.com>
Date: Fri Dec 20 23:54:23 2024 +0530

    SB Edited
Prajwals-MacBook-Air:CSBS admin$ git log
commit dea43c964d77b7e50c24a15c42428505b21531ce (HEAD -> new)
Author: CSBS 2023 <CSBSVTU2023@gmail.com>
Date: Fri Dec 20 23:54:23 2024 +0530

    SB Edited

commit 7c5187a42aba4dd0c1b8bf38beba3400544f987e
Author: Prajwal <Xyz@gmail.com>
Date: Thu Dec 19 01:41:44 2024 +0530

    File created
Prajwals-MacBook-Air:CSBS admin$
```

## PROGRAM 11

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

**STEP 1:** `git log -n`:

This command is used to view the most recent `n` commits in your Git repository. The `-n` flag specifies the number of commits to display. Example: `git log -5`

This command will display the last 5 commits in your current branch. Analyzing Git History with `git log -n`

- \* **Quick Overview:** You can quickly get a snapshot of recent changes without scrolling through a long list of commits.

- \* **Identifying Recent Issues:** By looking at the commit messages, you can identify recent bug fixes or feature additions.

- \* **Reviewing Code Changes:** You can use `git show <commit-hash>` to view the specific changes introduced by a commit. Changing Git History with `git log -n` While `git log -n` is primarily used for analyzing history, you can use it indirectly to change Git history. For example:

- \* **Identifying Commits for Revert or Cherry-Pick:** You can use this command to find specific commits that you want to revert or cherry-pick to another branch.

- \* **Reviewing Code Changes Before Rebase:** You can use `git log -n` to review the commits you plan to rebase onto another branch.

## OUTPUT:

```
bash CSBS X
Prajwals-MacBook-Air:CSBS admin$ git log -3
commit dea43c964d77b7e50c24a15c42428505b21531ce (HEAD -> new)
Author: CSBS 2023 <CSBSVTU2023@gmail.com>
Date: Fri Dec 20 23:54:23 2024 +0530

    SB Edited

commit 7c5187a42aba4dd0c1b8bf38beba3400544f987e
Author: Prajwal <Xyz@gmail.com>
Date: Thu Dec 19 01:41:44 2024 +0530

    File created

Prajwals-MacBook-Air:CSBS admin$ git log -2
commit dea43c964d77b7e50c24a15c42428505b21531ce (HEAD -> new)
Author: CSBS 2023 <CSBSVTU2023@gmail.com>
Date: Fri Dec 20 23:54:23 2024 +0530

    SB Edited

commit 7c5187a42aba4dd0c1b8bf38beba3400544f987e
Author: Prajwal <Xyz@gmail.com>
Date: Thu Dec 19 01:41:44 2024 +0530

    File created

Prajwals-MacBook-Air:CSBS admin$
```

## PROGRAM 12

**Write the commands to undo the changes introduced by the commit with the ID “abc123”.**

**STEP 1:** `git revert <COMMIT ID>`:

This Git command is used to reverse the changes introduced by a specific commit. It creates a new commit that undoes the changes made by the original commit. How it works:

- \* **Identify the Commit:** You need to find the commit ID of the commit you want to revert. You can use `git log` to find it.
- \* **Execute the Command:** Run the `git revert <COMMIT ID>` command, replacing `<COMMIT ID>` with the actual ID of the commit.
- \* **Create a New Commit:** Git creates a new commit that reverses the changes. The commit message will typically include information about the original commit and the revert.

**Key Points:**

- \* **Preserves History:** Revert creates a new commit, preserving the original commit in the history. This is useful for undoing mistakes or reverting unwanted changes.
- \* **Resolves Conflicts:** If there are conflicts between the revert and the current state of the branch, Git will prompt you to resolve them manually.
- \* **Multiple Reverts:** You can revert multiple commits by providing their respective IDs.

Example:

```
git revert 1234567890abcdef
```

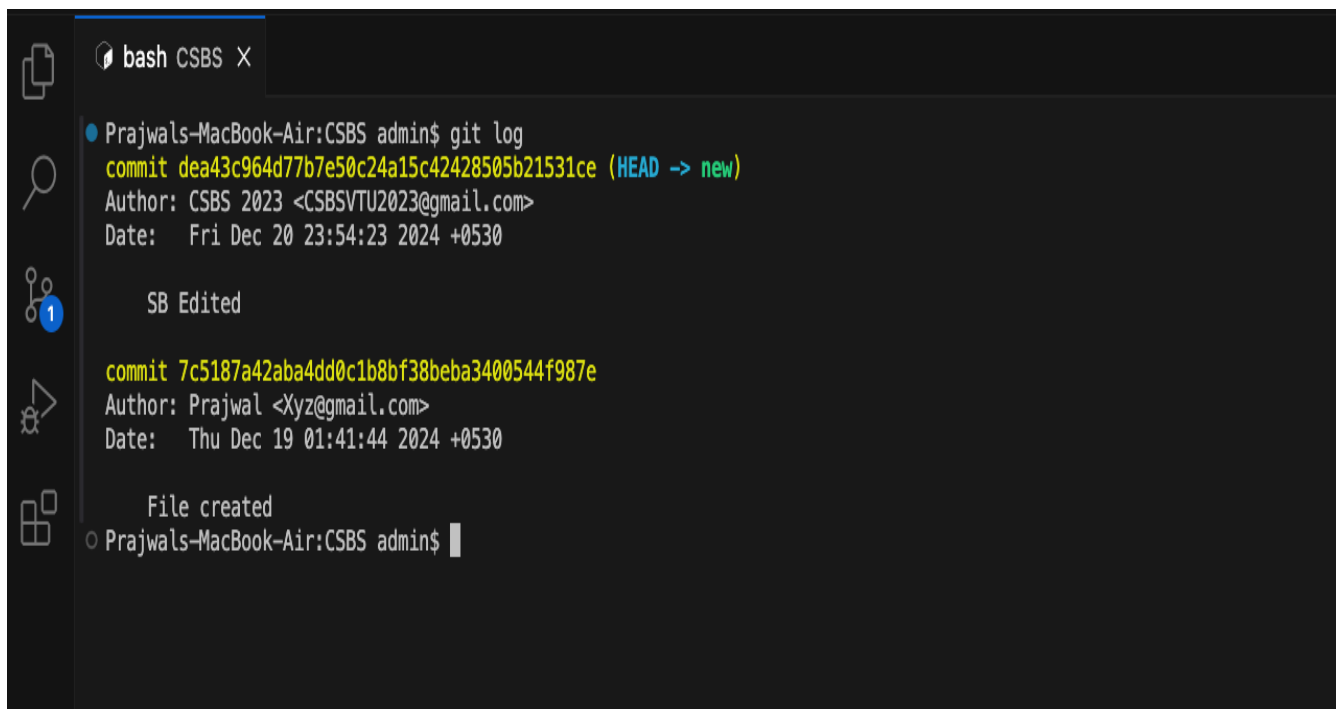
This command will revert the commit with the ID 1234567890abcdef.

**Analyzing and Changing Git History with `git revert`**

- \* **Undoing Mistakes:** If you realize that a previous commit introduced a bug or unwanted changes, you can use `git revert` to undo those changes.
  - \* **Reverting Specific Changes:** You can revert specific commits without affecting other changes in the same branch.
  - \* **Creating a Clean History:** By carefully using `git revert`, you can keep your Git history clean and understandable.
- Important Considerations:**
- \* **Careful Usage:** Revert can sometimes introduce unintended side effects, especially if the reverted commit is part of a complex series of changes.

- \* Use with Caution: Always back up your repository before making significant changes to the history.
- \* Consider Alternatives: If you need to modify the history more extensively, consider using Git's interactive rebase or reset commands. By understanding git revert, you can effectively manage your Git history and undo changes when necessary.

## OUTPUT:



```
bash CSBS X
Prajwals-MacBook-Air:CSBS admin$ git log
commit dea43c964d77b7e50c24a15c42428505b21531ce (HEAD -> new)
Author: CSBS 2023 <CSBSVTU2023@gmail.com>
Date: Fri Dec 20 23:54:23 2024 +0530

    SB Edited

commit 7c5187a42aba4dd0c1b8bf38beba3400544f987e
Author: Prajwal <Xyz@gmail.com>
Date: Thu Dec 19 01:41:44 2024 +0530

    File created
Prajwals-MacBook-Air:CSBS admin$
```