VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belagavi-560014,Karnataka

GIT LABORATORY PROGRAMS REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR PROJECT MANAGEMENT WITH GIT SUBJECT (BCS358C)

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

SubmittedBy

NAME: HARSHITHA V

USN: 1SV22CS041

Under the guidance of

Prof. Merlin B

Assistant Professor,

Dept of ISE

Department of Computer Science and Engineering

SHRIDEVI INSTITUTEOFENGINEERINGANDTECHNOLOGY

(Affiliated To Visvesvaraya Technological University) Sira Road, Tumakuru–572106, Karnataka.

2023-2024

Sri Shridevi Charitable Trust (R.)

SHRIDEVI INSTITUTE OF ENGINEERING AND TECHNOLOGY

Sira Road, Tumkur - 572 106, Karnataka, India.

Phone: 0816 - 2212629 | Principal: 0816 - 2212627, 9686114899 | Telefax: 0816 - 2212628

Email: info@shrideviengineering.org, principal@shrideviengineering.org | Website: www.shrideviengineering.org

(Approved by AICTE, New Delhi, Recognised by Govt. of Karnataka and Affiliated to Visvesvaraya Technological University, Belagavi)

SHRIDEVI EDUCATION

ESTD:2002 20 YEARS Academic Excellence

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that, Git Lab Programs has been successfully carried out by HARSHITHAV[1SV22CS041] in partial fulfillment for the PROJECT MANAGEMENT WITH GIT (BCS358C) Subject of **Bachelor of Engineering in Computer Science and Engineering Department** of the **Visvesvaraya Technological University, Belagavi** during the academic year **2023-24.** It is certified that all the corrections/suggestions indicated for internal assessments have been incorporated in the report. The Git Lab Programs has been approved as it certifies the academic requirements in respect of PROJECT MANAGEMENT WITH GIT (BCS358C) Subject of Bachelor of Engineering Degree.

Signature of Lab Coordinator

MRS MERLIN. B.BE.,MTech.,

Assistant Professor,
Dept of ISE, SIET, Tumakuru

Signature of H.O.D

DR. BASAVESHA D.BE.,MTech.,PhD,

Associate Professor & HOD,
Dept of CSE, SIET, Tumakuru.

Name of the Examiners	Signature with date
1	....
2..................................	....

Sri Shridevi Charitable Trust (R.)
SHRIDEVI INSTITUTE OF ENGINEERING AND TECHNOLOGY
Sira Road, Tumkur - 572 106, Karnataka, India.
Phone: 0816 - 2212629 | Principal: 0816 - 2212627, 9686114899 | Telefax: 0816 - 2212628
Email: info@shrideviengineering.org, principal@shrideviengineering.org | Website: www.shrideviengineering.org
(Approved by AICTE, New Delhi, Recognised by Govt. of Karnataka and Affiliated to Visvesvaraya Technological University, Belagavi)
SHRIDEVI EDUCATION
ESTD:2002 20 YEARS Academic Excellence

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PARTICULARS OF THE EXPERIMENTS PERFORMED
CONTENTS
PROJECT MANAGEMENT WITH GIT

EXPT NO	DATE	TITLE OF THE EXPT	CONDUCTION (20M)	VIVA (10M)	TOTAL (30M)	SIGN
1	22/11/23	Setting up basic commands				
2	29/12/23	Creating and managing branches				
3	6/12/23	Creating and managing branches				
4	20/12/23	Collaboration and remote repositories				
5	27/12/23	Collaboration and remote repositories				
6	10/12/24	Collaboration and remote repositories				
7	10/1/24	Git tags and releases				
8	17/1/24	Advanced git operations				
9	31/1/24	Analysing and changing git history				
10	14/2/24	Analysing and changing git history				
11	21/2/24	Analysing and changing git history				
12	28/2/24	Analysing and changing git history				
		AVERAGE				

SIGNATURE OF COURSE INSTRUCTOR

1. ______________________________

2. ______________________________

EXPERIMENT NO-01

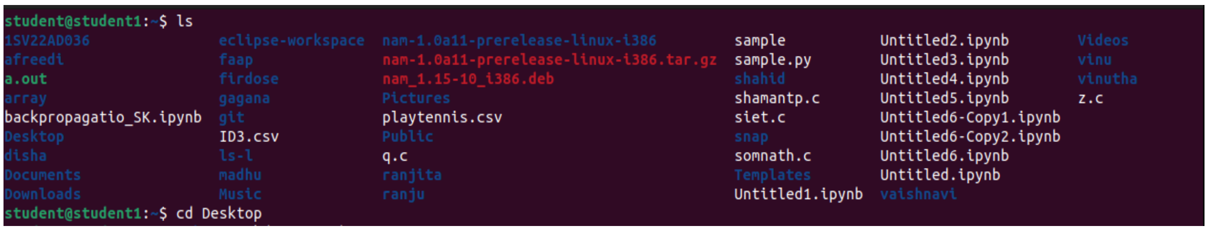
Setting up basic commands:

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.

The Basic Commands that are used here are:

* $ ls :this command shows the list of files and folders present in the system.

* $cd: Desktop – cd(change directory),this command is used to change the present Directory into Desktop.

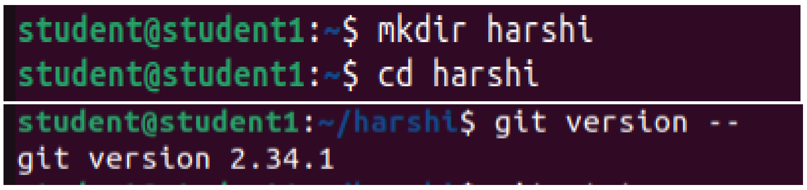


```
student@student1:~$ ls
1SV22AD036               eclipse-workspace  nam-1.0a11-prerelease-linux-i386         sample           Untitled2.ipynb          Videos
afreedi                  faap               nam-1.0a11-prerelease-linux-i386.tar.gz  sample.py        Untitled3.ipynb          vinu
a.out                    firdose            nam_1.15-10_i386.deb                     shahid           Untitled4.ipynb          vinutha
array                    gagana             Pictures                                 shamantp.c       Untitled5.ipynb          z.c
backpropagatio_SK.ipynb  git                playtennis.csv                           siet.c           Untitled6-Copy1.ipynb
Desktop                  ID3.csv            Public                                   snap             Untitled6-Copy2.ipynb
disha                    ls-l               q.c                                      somnath.c        Untitled6.ipynb
Documents                madhu              ranjita                                  Templates        Untitled.ipynb
Downloads                Music              ranju                                    Untitled1.ipynb  vaishnavi
student@student1:~$ cd Desktop
```

*$ mkdir harshi: mkdir(make directory),here new directory is created which is named as harshi.

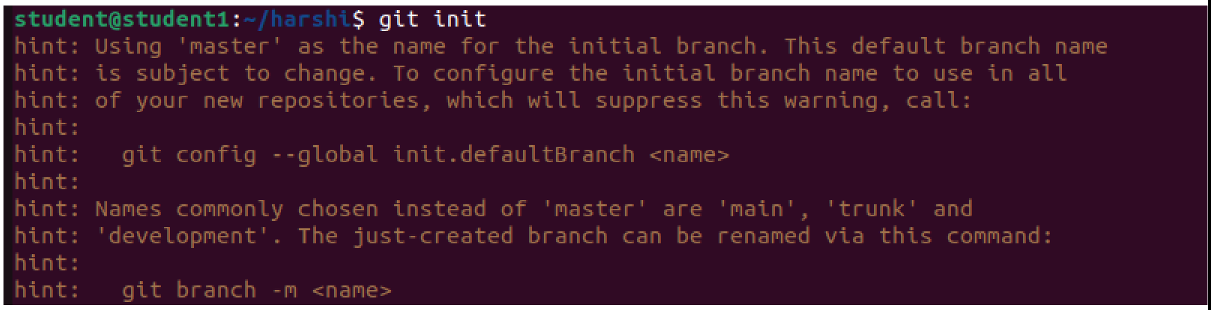
*$ cd harshi: command is used to change the current working directory into a new directory named as harshi.

*$ git –version: this command is used to check whether git package is installed and also to know the version.



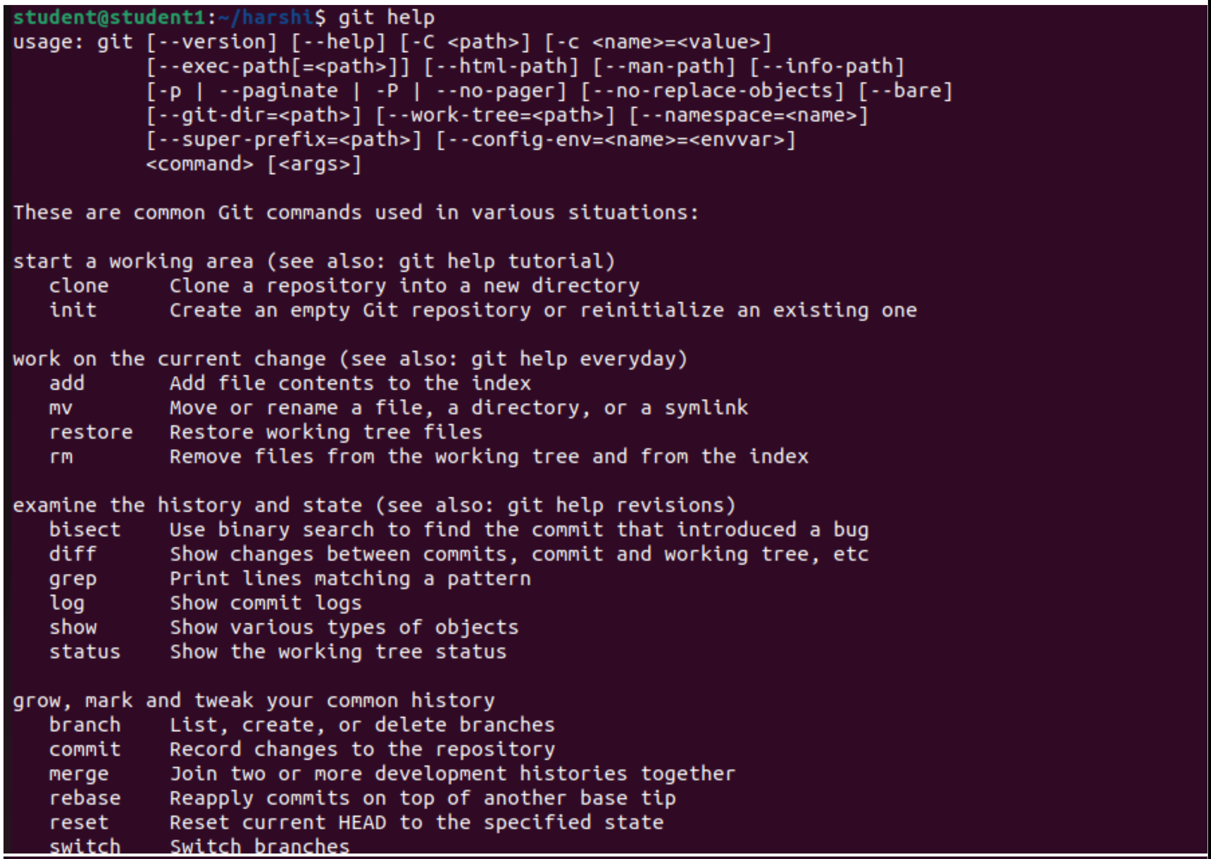
```
student@student1:~$ mkdir harshi
student@student1:~$ cd harshi
student@student1:~/harshi$ git version --
git version 2.34.1
```

*$ git init : to initialize a new git repository int the current directory. When you run this command in a directory,Git creates a new subdirectory named '.git' that contains all of the necessary metadata for the repository. This '.git' directory is where Git stores information about the repository's configuration,commits,branches and more. We can start adding the files, making commits,and managing our version controlled project using Git.

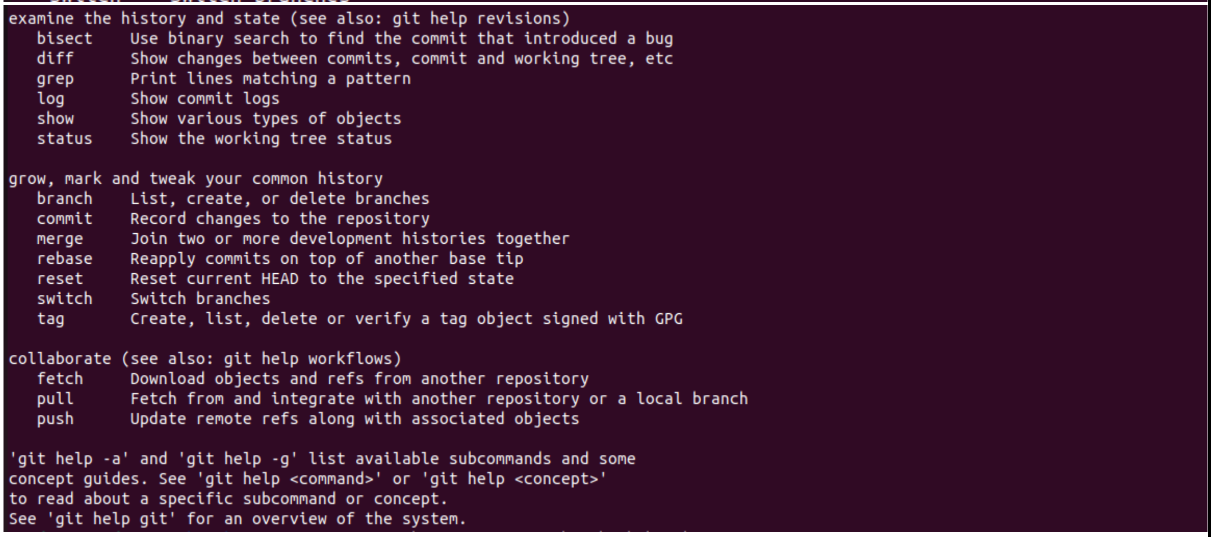


```
student@student1:~/harshi$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global 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>
```

*$ git help : this command is used to access the Git manual and get help on various Git commands and topics.you can use it in combination with a specific Git command to get detailed information about the command. For eg,$ git help command.



```
student@student1:~/harshi$ git help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone     Clone a repository into a new directory
   init      Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add       Add file contents to the index
   mv        Move or rename a file, a directory, or a symlink
   restore   Restore working tree files
   rm        Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect    Use binary search to find the commit that introduced a bug
   diff      Show changes between commits, commit and working tree, etc
   grep      Print lines matching a pattern
   log       Show commit logs
   show      Show various types of objects
   status    Show the working tree status

grow, mark and tweak your common history
   branch    List, create, or delete branches
   commit    Record changes to the repository
   merge     Join two or more development histories together
   rebase    Reapply commits on top of another base tip
   reset     Reset current HEAD to the specified state
   switch    Switch branches
```



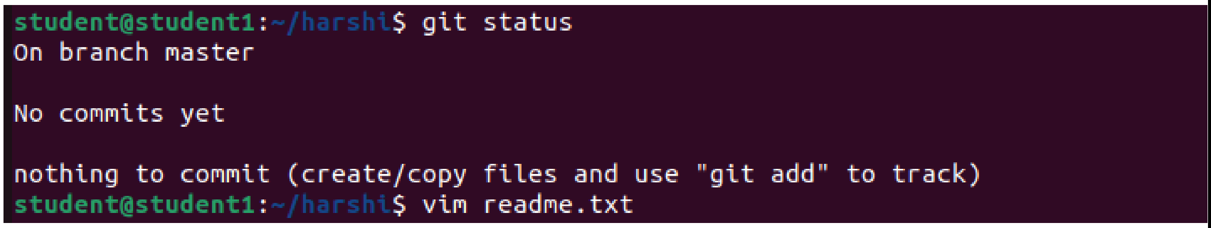
```
examine the history and state (see also: git help revisions)
   bisect    Use binary search to find the commit that introduced a bug
   diff      Show changes between commits, commit and working tree, etc
   grep      Print lines matching a pattern
   log       Show commit logs
   show      Show various types of objects
   status    Show the working tree status

grow, mark and tweak your common history
   branch    List, create, or delete branches
   commit    Record changes to the repository
   merge     Join two or more development histories together
   rebase    Reapply commits on top of another base tip
   reset     Reset current HEAD to the specified state
   switch    Switch branches
   tag       Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch     Download objects and refs from another repository
   pull      Fetch from and integrate with another repository or a local branch
   push      Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
```

*$ git status : It’s a fundamental Git command used to display the state of working directory and the staging area. When you run ‘git status’, Git will show you:

- Which files are staged for commit in the staging area.
- Which files are modified but not yet staged.
- Which files are untracked
- Information about the current branch,such as whether your branch is ahead or behind its remote counterpart.

This command is extremely useful for understanding what changes have been made and what actions need to be taken before pushing changes to a remote repository like GitHub. It helps us to manage our repository effectively and keep track of our progress.

*$ vim or vi filename.txt : command is used to open a new file and the file name is readme.txt .

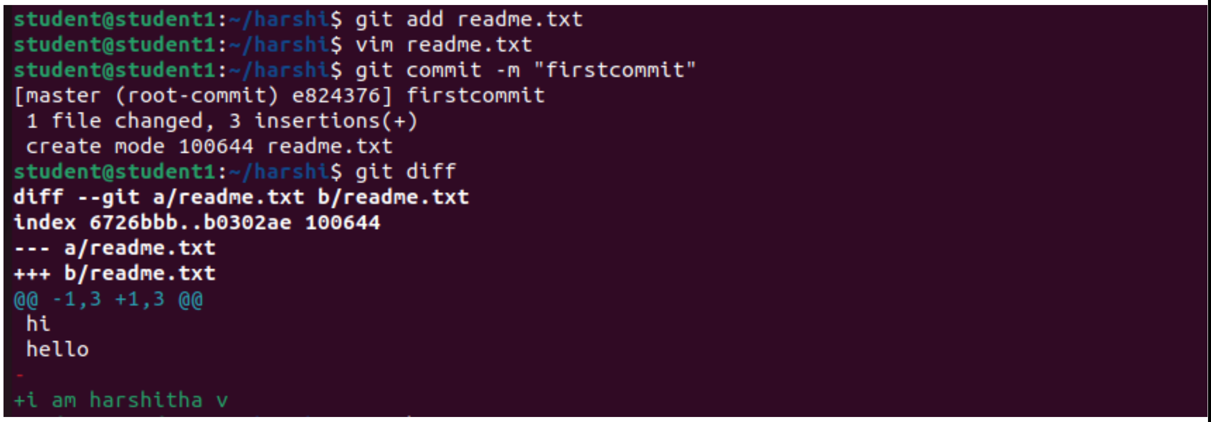


```
student@student1:~/harshi$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
student@student1:~/harshi$ vim readme.txt
```

*$ git add filename.txt : command is used to stage changes made to the specified file named filename.txt for the next commit in your Git repository.

When you make changes to files in your working directory, Git initially considers them as modified but not yet staged for commit. By running git add filename.txt, you inform Git that you want to include the changes in filename.txt in the next commit. This action moves the changes to the staging area, preparing them to be committed.

*$ git commit –m “commit message”: command is used to commit staged changes to your Git repository along with a commit message provided inline using the -m flag. After running the git commit command, Git will create a new commit with the staged changes and associate the provided commit message with it. This helps maintain a clear history of changes in your Git repository.



```
student@student1:~/harshi$ git add readme.txt
student@student1:~/harshi$ vim readme.txt
student@student1:~/harshi$ git commit -m "firstcommit"
[master (root-commit) e824376] firstcommit
 1 file changed, 3 insertions(+)
 create mode 100644 readme.txt
student@student1:~/harshi$ git diff
diff --git a/readme.txt b/readme.txt
index 6726bbb..b0302ae 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,3 +1,3 @@
 hi
 hello
-
+i am harshitha v
```

*$ git diff: git diff is a multi-use Git command that when executed runs a diff function on Git data sources. These data sources can be commits, branches, files and more.

* $ git config --global user.email “your email@example.com”: command s used to set or update the global Git email configuration on your system. This command is typically used once to configure your email address globally, so you don't have to specify it every time you make a commit.

Replace "your_email@example.com" with your actual email address. For example:

$ git config --global user.email “harshitha@gmail.com”

By setting your email address globally, Git will use this email for all repositories on your system unless overridden by a local configuration specific to a particular repository. This helps associate your commits with your email address, providing contact information for collaborators and maintaining a clear history of changes.

* $ git config --global user.name “your username”: command is used to set or update the global Git username configuration on your system. This command is typically used once to configure your username globally, so you don't have to specify it every time you make a commit.

Replace "Your Username" with your actual Git username. For example:

$ git config –global user.name “harshitha”

By setting your username globally, Git will use this username for all repositories on your system unless overridden by a local configuration specific to a particular repository. This helps identify who made each commit in the repository's history.

* $ git remote add origin “remote repository URL”: command is used to add a remote repository URL to your local Git repository with the name "origin."

This command establishes a connection between your local repository and a remote repository hosted on a server, such as GitHub or GitLab. The term "origin" is a conventionally used name for the default remote repository, but you can choose any name you prefer.

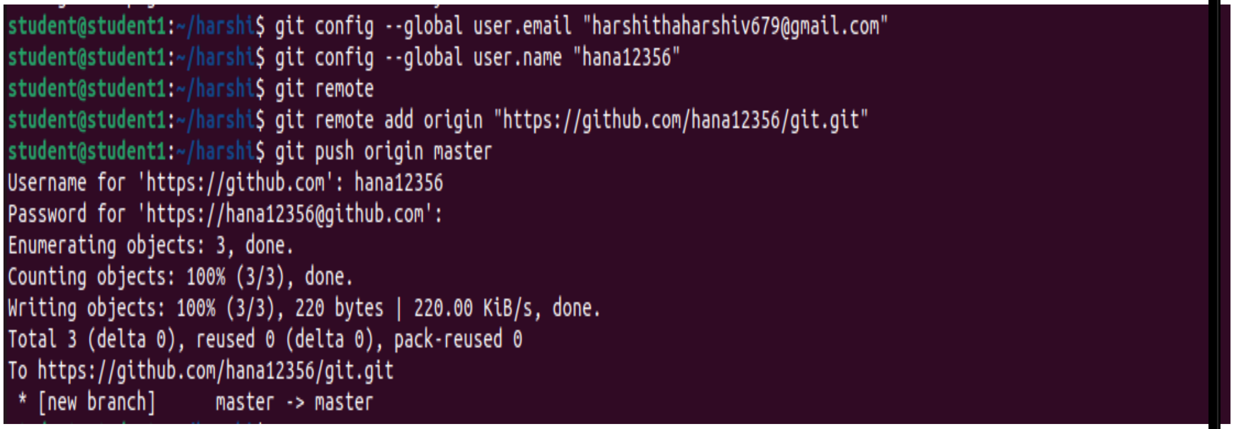
* $ git push origin master :The command git push origin master is used to push the commits from your local master branch to the remote repository named origin.

Here's a breakdown of what each part of the command does:

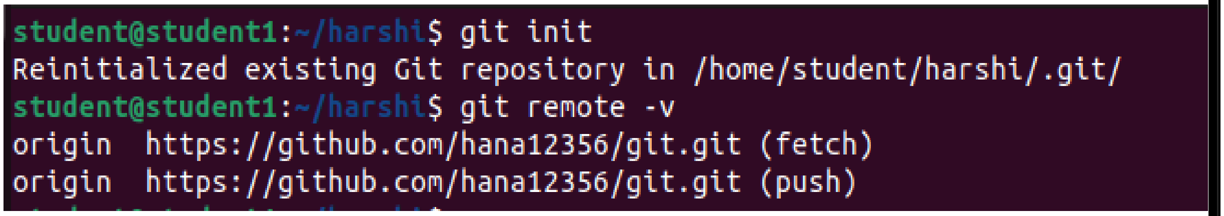
- **git push**: This is the Git command used to push commits from your local repository to a remote repository.
- **origin**: This refers to the name of the remote repository you're pushing to. In Git terminology, "origin" is a common name used to refer to the default remote repository.
- **master**: This refers to the local branch that you're pushing. In Git, "master" is the default name for the main branch of a repository.

So,when you run git push origin master, you're telling Git to push the commits from your local master branch to the remote repository named origin.

* $ git remote –v : command is used to view the list of remote repositories associated with your local Git repository along with their corresponding URLs. When you run this command, Git will display a list of remote repositories and their corresponding fetch and push URLs.



```
student@student1:~/harshi$ git config --global user.email "harshithaharshiv679@gmail.com"
student@student1:~/harshi$ git config --global user.name "hana12356"
student@student1:~/harshi$ git remote
student@student1:~/harshi$ git remote add origin "https://github.com/hana12356/git.git"
student@student1:~/harshi$ git push origin master
Username for 'https://github.com': hana12356
Password for 'https://hana12356@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 220 bytes | 220.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/hana12356/git.git
 * [new branch]      master -> master
```



```
student@student1:~/harshi$ git init
Reinitialized existing Git repository in /home/student/harshi/.git/
student@student1:~/harshi$ git remote -v
origin  https://github.com/hana12356/git.git (fetch)
origin  https://github.com/hana12356/git.git (push)
```

EXPERIMENT NO -02

Creating and Managing Branches

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

*$ git branch feature-branch : command is used to create a new branch named featurebranch in your Git repository. After running this command, you'll have a new branch based on your current branch's state.

This command will create a new branch named feature-branch at your current commit. However, it won't switch you to that branch automatically. To start working on the new branch, you need to check it out using git checkout or git switch.

* $ git checkout feature-branch : This command will switch your working directory to the feature-branch branch.

* $ vi branchfile.txt : The command vi branchfile.txt opens the file named branchfile.txt in the Vim text editor.

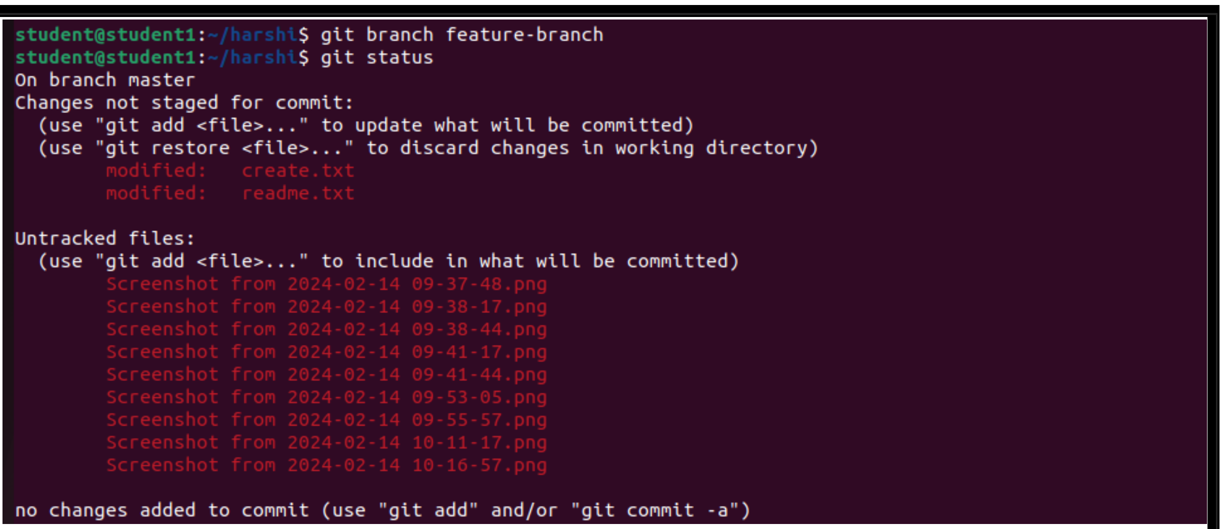
When you run vi branchfile.txt, Vim will open the file in its default mode, which is usually the command mode. From there, you can navigate, edit, and save the file using various keyboard shortcuts and commands.

*$ git add branchfile.txt : command stages the changes made to the file named branchfile.txt for the next commit in your Git repository. This means that Git will track the changes made to this file when you commit them.

*$ git log --oneline –decorate : used to display a compact and decorated version of the commit history in your Git repository.

Here's what each option does:

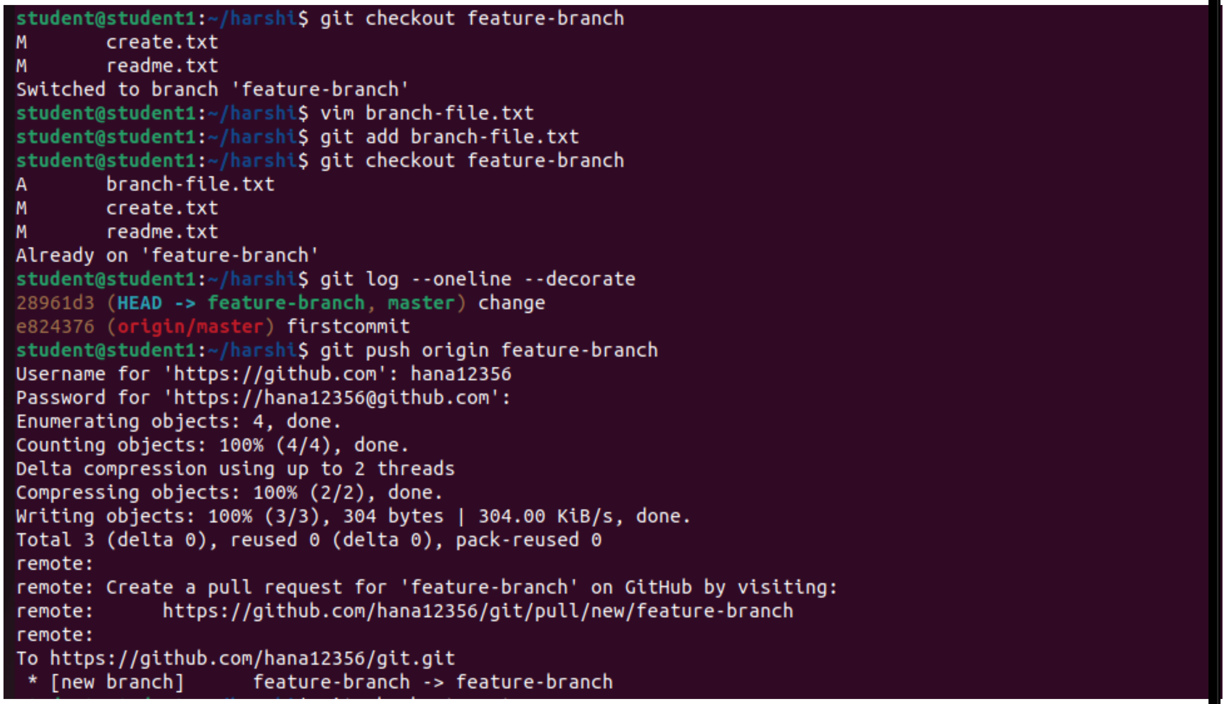
- **--oneline**: This option displays each commit as a single line, showing only the first line of the commit message along with the commit hash.
- **--decorate**: This option annotates the output with additional information, such as branch and tag names that point to each commit.



```
student@student1:~/harshi$ git branch feature-branch
student@student1:~/harshi$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   create.txt
        modified:   readme.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Screenshot from 2024-02-14 09-37-48.png
        Screenshot from 2024-02-14 09-38-17.png
        Screenshot from 2024-02-14 09-38-44.png
        Screenshot from 2024-02-14 09-41-17.png
        Screenshot from 2024-02-14 09-41-44.png
        Screenshot from 2024-02-14 09-53-05.png
        Screenshot from 2024-02-14 09-55-57.png
        Screenshot from 2024-02-14 10-11-17.png
        Screenshot from 2024-02-14 10-16-57.png

no changes added to commit (use "git add" and/or "git commit -a")
```

***$git status** : It’s a fundamental Git command used to display the state of working directory and the staging area. When you run ‘git status’, Git will show you:

- ➢ Which files are staged for commit in the staging area.
- ➢ Which files are modified but not yet staged.
- ➢ Which files are untracked
- ➢ Information about the current branch,such as whether your branch is ahead or behind its remote counterpart.



```
student@student1:~/harshi$ git checkout feature-branch
M       create.txt
M       readme.txt
Switched to branch 'feature-branch'
student@student1:~/harshi$ vim branch-file.txt
student@student1:~/harshi$ git add branch-file.txt
student@student1:~/harshi$ git checkout feature-branch
A       branch-file.txt
M       create.txt
M       readme.txt
Already on 'feature-branch'
student@student1:~/harshi$ git log --oneline --decorate
28961d3 (HEAD -> feature-branch, master) change
e824376 (origin/master) firstcommit
student@student1:~/harshi$ git push origin feature-branch
Username for 'https://github.com': hana12356
Password for 'https://hana12356@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), 304 bytes | 304.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature-branch' on GitHub by visiting:
remote:      https://github.com/hana12356/git/pull/new/feature-branch
remote:
To https://github.com/hana12356/git.git
 * [new branch]      feature-branch -> feature-branch
```

*$ git push origin feature-branch : used to push the commits from your local featurebranch to the remote repository named origin. This is typically done when you want to share your changes with others or synchronize your work between your local repository and the remote repository.

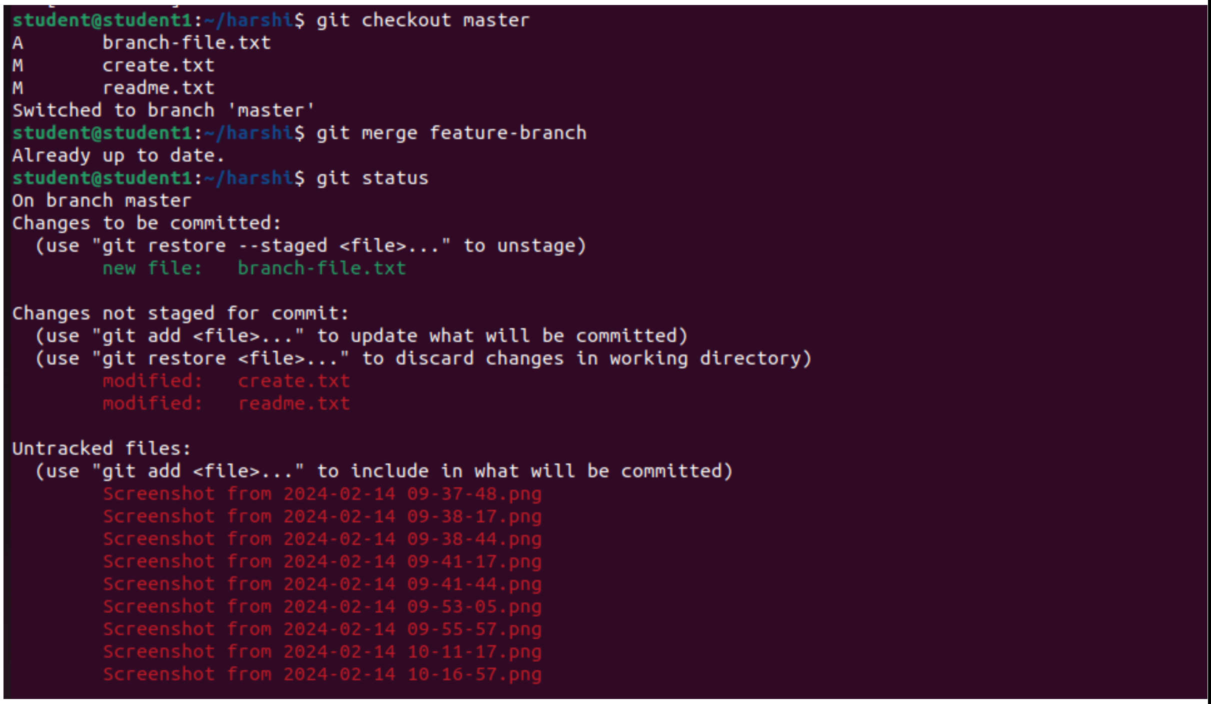
Here's a breakdown of what each part of the command does:

- ➢ **git push**: This is the Git command used to push commits from your local repository to a remote repository.

- **origin**: This refers to the name of the remote repository you're pushing to. In Git terminology, "origin" is a common name used to refer to the default remote repository.
- **feature-branch**: This is the name of the local branch you want to push. It's assumed that you've already created this branch locally and made some commits on it.

*$ git checkout master : used to switch to the master branch in your Git repository.When you run this command, Git updates your working directory to reflect the state of the master branch.

This means that any changes you make or files you create or modify will be based on the master branch. After running this command, you'll be on the master branch, and you can start working on it, making changes,creating commits etc.



```
student@student1:~/harshit$ git checkout master
A       branch-file.txt
M       create.txt
M       readme.txt
Switched to branch 'master'
student@student1:~/harshit$ git merge feature-branch
Already up to date.
student@student1:~/harshit$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branch-file.txt

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Screenshot from 2024-02-14 09-37-48.png
        Screenshot from 2024-02-14 09-38-17.png
        Screenshot from 2024-02-14 09-38-44.png
        Screenshot from 2024-02-14 09-41-17.png
        Screenshot from 2024-02-14 09-41-44.png
        Screenshot from 2024-02-14 09-53-05.png
        Screenshot from 2024-02-14 09-55-57.png
        Screenshot from 2024-02-14 10-11-17.png
        Screenshot from 2024-02-14 10-16-57.png
```

$ git merge feature-branch : command is used to merge changes from the specified branch (in this case, feature-branch) into the current branch. Typically, you execute this command while you're on the branch where you want to merge the changes.

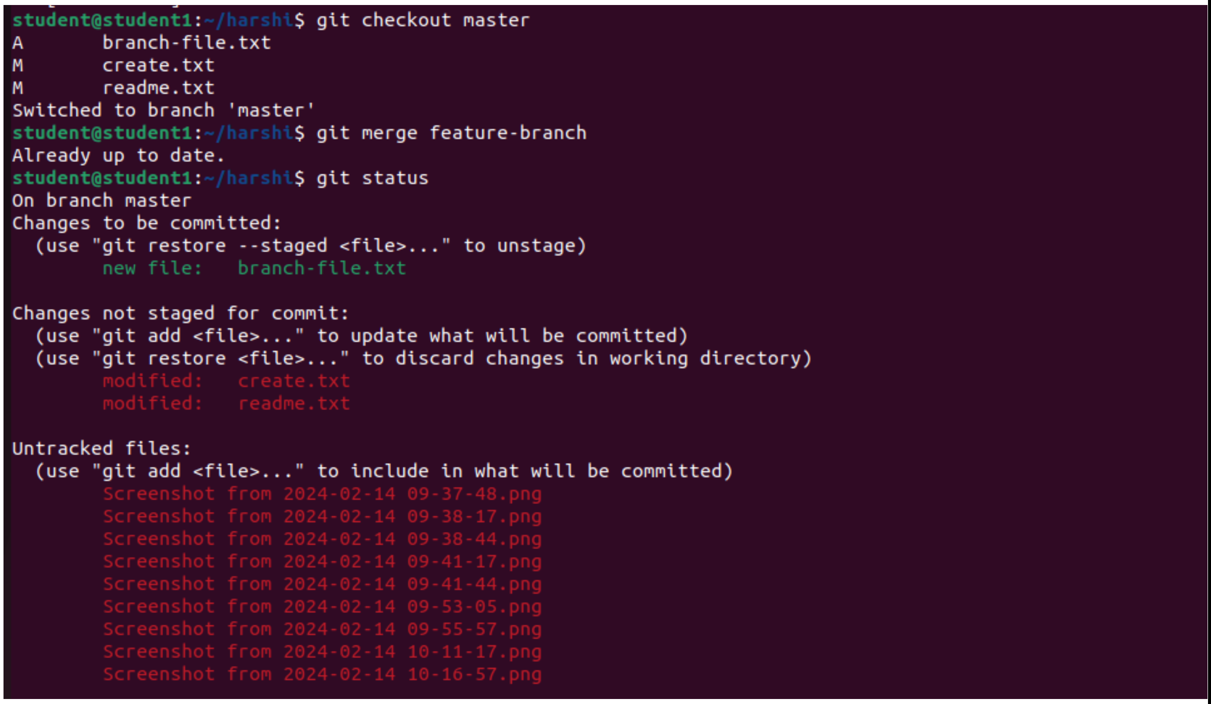
This command will incorporate the changes from featurebranch into the branch you're currently on.After successfully merging feature-branch into master, you'll have all the changes from feature-branch incorporated into master, and you can continue working on master with the merged changes.

*$ git push origin master : used to push the commits from your local master branch to the remote repository named origin. This is a common command used to update the remote repository with the changes you've made locally. Here's a breakdown of what each part of the command does:

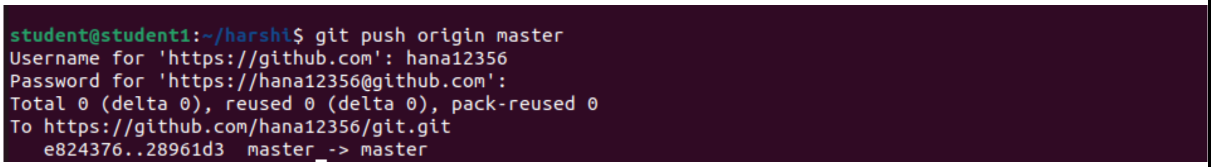
> **git push**: This is the Git command used to push commits from your local repository to a remote repository.

> **origin**: This refers to the name of the remote repository you're pushing to. In Git terminology, "origin" is a common name used to refer to the default remote repository.

> **master**: This is the name of the local branch you're pushing. In many Git repositories, master is the default name for the main branch.



```
student@student1:~/harshi$ git checkout master
A       branch-file.txt
M       create.txt
M       readme.txt
Switched to branch 'master'
student@student1:~/harshi$ git merge feature-branch
Already up to date.
student@student1:~/harshi$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branch-file.txt

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Screenshot from 2024-02-14 09-37-48.png
        Screenshot from 2024-02-14 09-38-17.png
        Screenshot from 2024-02-14 09-38-44.png
        Screenshot from 2024-02-14 09-41-17.png
        Screenshot from 2024-02-14 09-41-44.png
        Screenshot from 2024-02-14 09-53-05.png
        Screenshot from 2024-02-14 09-55-57.png
        Screenshot from 2024-02-14 10-11-17.png
        Screenshot from 2024-02-14 10-16-57.png
```



```
student@student1:~/harshi$ git push origin master
Username for 'https://github.com': hana12356
Password for 'https://hana12356@github.com':
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/hana12356/git.git
   e824376..28961d3  master -> master
```

EXPERIMENT NO -03

Creating and Managing Branches

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

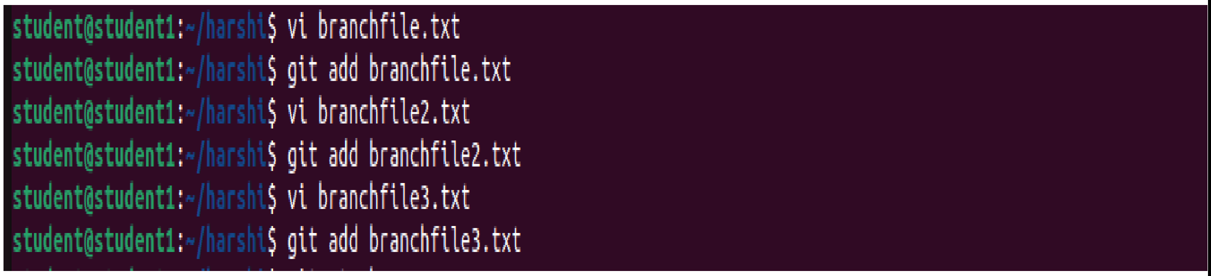
*$ git stash : command is used to temporarily save changes in your working directory and staging area so that you can work on something else or switch branches without committing them.

When you run git stash, Git will save your changes into a stack of stashes, leaving your working directory and staging area clean. You can then switch branches or perform other operations without worrying about the changes you've stashed.

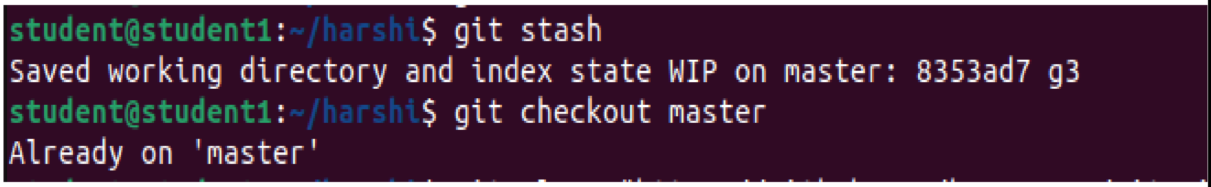
*$ git stash apply : used to retrieve and reapply the most recent stash from the stash stack onto your current working directory. This command will reapply the changes from the stash onto your working directory without removing the stash from the stack.

*$ git stash list : used to display the list of stashes in your Git repository's stash stack. It shows all the stashes you've created, along with a reference for each stash.

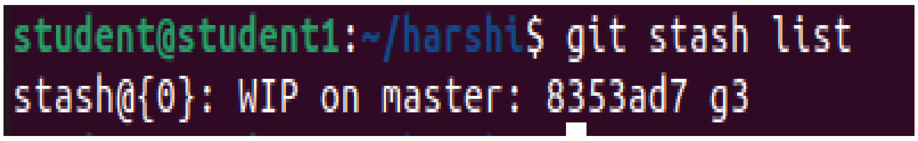
When you run this command, Git will list all the stashes you've created in the repository. Each stash will be listed along with a reference, typically in the format stash@{n}, where n is the index of the stash in the stash .



```
student@student1:~/harshi$ vi branchfile.txt
student@student1:~/harshi$ git add branchfile.txt
student@student1:~/harshi$ vi branchfile2.txt
student@student1:~/harshi$ git add branchfile2.txt
student@student1:~/harshi$ vi branchfile3.txt
student@student1:~/harshi$ git add branchfile3.txt
```



```
student@student1:~/harshi$ git stash
Saved working directory and index state WIP on master: 8353ad7 g3
student@student1:~/harshi$ git checkout master
Already on 'master'
```



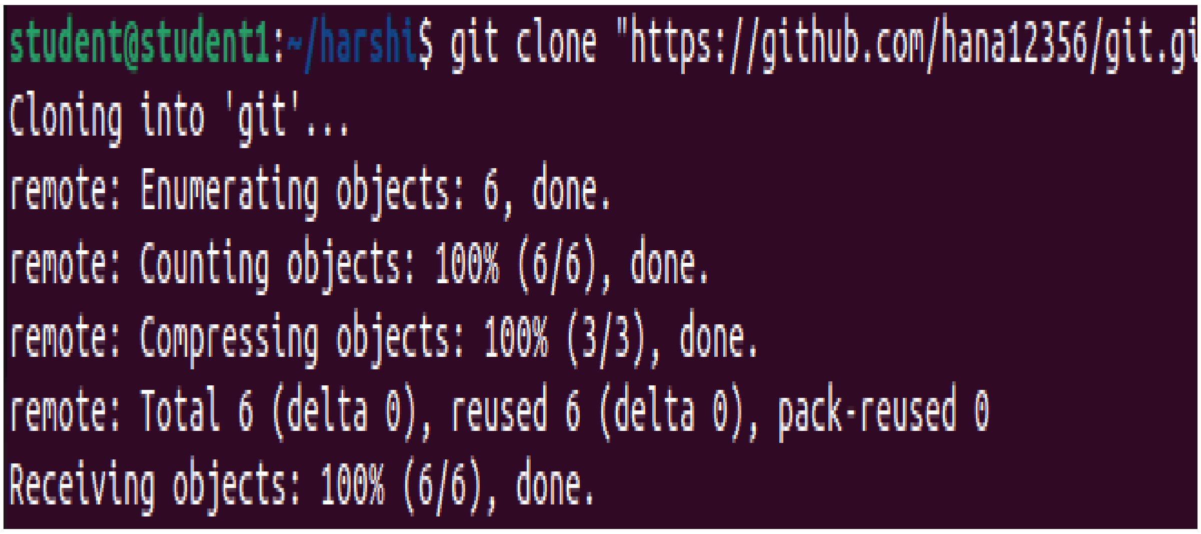
```
student@student1:~/harshi$ git stash list
stash@{0}: WIP on master: 8353ad7 g3
```

EXPERIMENT NO -04

Collaboration and Remote Repositories
Clone a remote Git repository to your local machine.

*$ git clone "repository URL" : The git clone command is used to create a copy of an existing Git repository in a new directory. This is useful when you want to start working on a project that already exists in a remote repository, such as on GitHub or GitLab.

*Repository URL is the URL of the remote repository you want to clone.
After cloning the repository, you'll have a complete copy of the project's history and files on your local machine. You can then make changes, create commits, and push them back to the remote repository as needed.



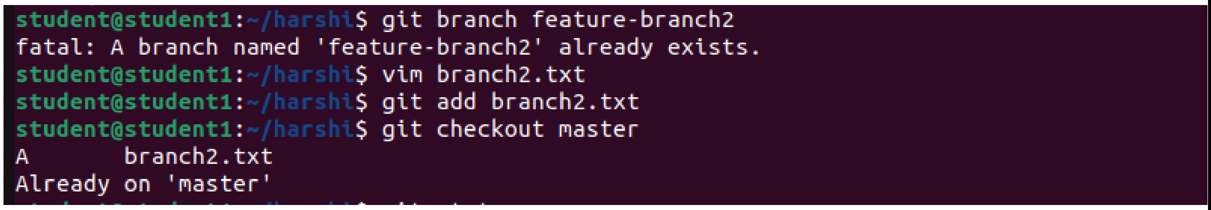
```
student@student1:~/harshi$ git clone "https://github.com/hana12356/git.gi
Cloning into 'git'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.
```

EXPERIMENT NO -05

Collaboration and Remote Repositories

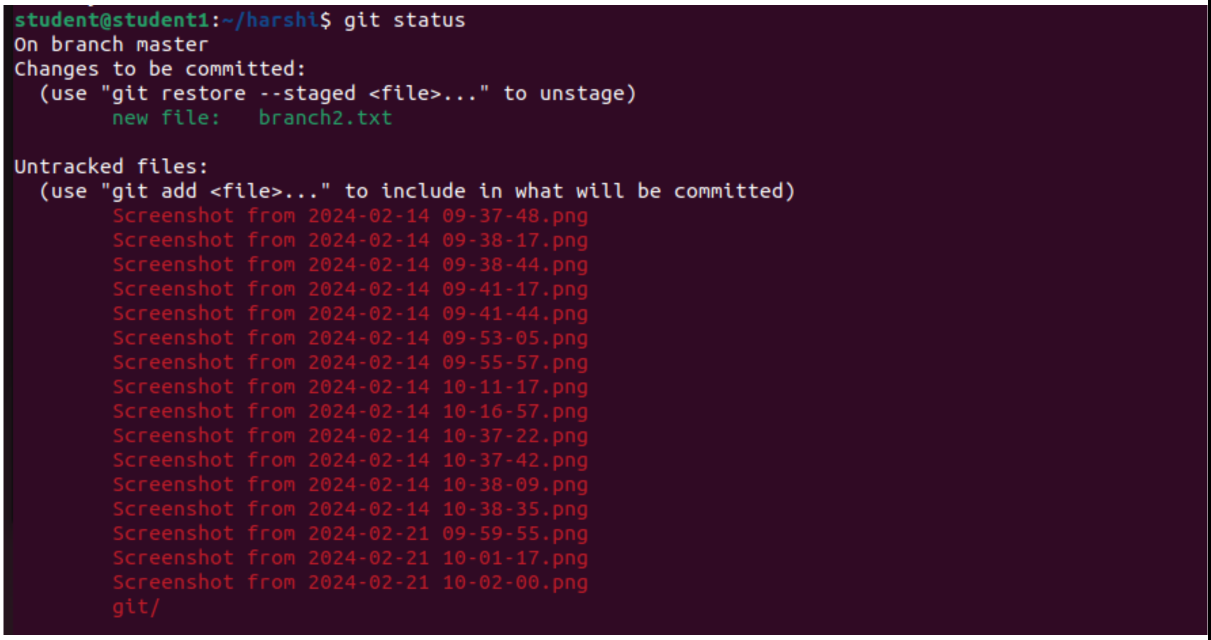
Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

***Rebasing**: It is changing the base of your branch from one commit to another commit making it appear as if you had created a branch froma different commit.



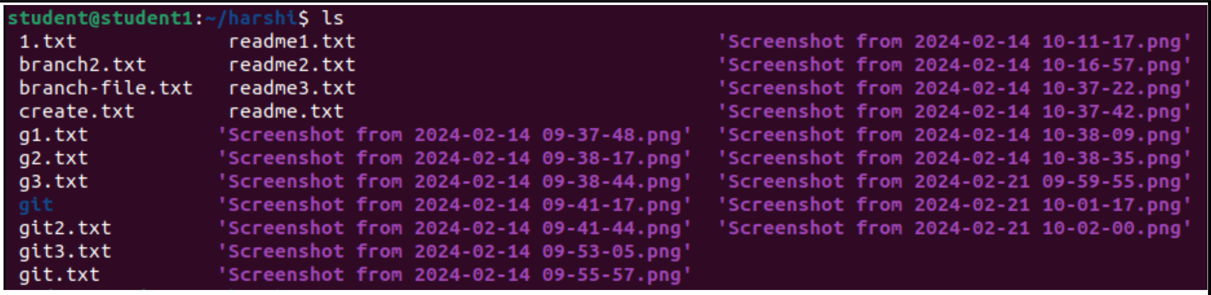
```
student@student1:~/harshi$ git branch feature-branch2
fatal: A branch named 'feature-branch2' already exists.
student@student1:~/harshi$ vim branch2.txt
student@student1:~/harshi$ git add branch2.txt
student@student1:~/harshi$ git checkout master
A       branch2.txt
Already on 'master'
```

*$ git commit –m “commit message”: command is used to commit staged changes to your Git repository along with a commit message provided inline using the -m flag. After running the git commit command, Git will create a new commit with the staged changes and associate the provided commit message with it. This helps maintain a clear history of changes in your Git repository.

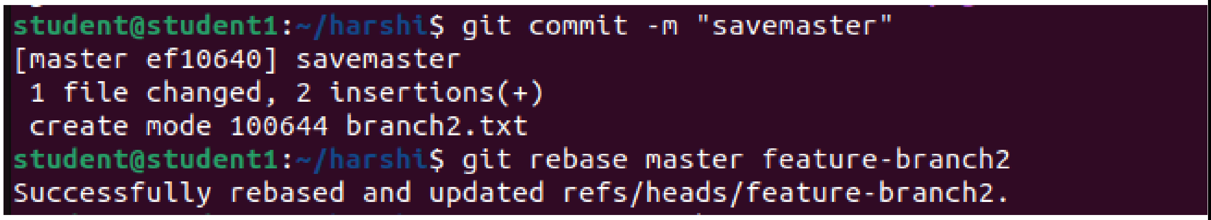


```
student@student1:~/harshi$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branch2.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Screenshot from 2024-02-14 09-37-48.png
        Screenshot from 2024-02-14 09-38-17.png
        Screenshot from 2024-02-14 09-38-44.png
        Screenshot from 2024-02-14 09-41-17.png
        Screenshot from 2024-02-14 09-41-44.png
        Screenshot from 2024-02-14 09-53-05.png
        Screenshot from 2024-02-14 09-55-57.png
        Screenshot from 2024-02-14 10-11-17.png
        Screenshot from 2024-02-14 10-16-57.png
        Screenshot from 2024-02-14 10-37-22.png
        Screenshot from 2024-02-14 10-37-42.png
        Screenshot from 2024-02-14 10-38-09.png
        Screenshot from 2024-02-14 10-38-35.png
        Screenshot from 2024-02-21 09-59-55.png
        Screenshot from 2024-02-21 10-01-17.png
        Screenshot from 2024-02-21 10-02-00.png
        git/
```

* $ git rebase master feature-branch : for rebasing, for apply git checkout master command to switch from feature branch to master branch and then apply git commit command and commit a message.



```
student@student1:~/harshi$ ls
 1.txt           readme1.txt                                  'Screenshot from 2024-02-14 10-11-17.png'
 branch2.txt     readme2.txt                                  'Screenshot from 2024-02-14 10-16-57.png'
 branch-file.txt readme3.txt                                  'Screenshot from 2024-02-14 10-37-22.png'
 create.txt      readme.txt                                   'Screenshot from 2024-02-14 10-37-42.png'
 g1.txt         'Screenshot from 2024-02-14 09-37-48.png'     'Screenshot from 2024-02-14 10-38-09.png'
 g2.txt         'Screenshot from 2024-02-14 09-38-17.png'     'Screenshot from 2024-02-14 10-38-35.png'
 g3.txt         'Screenshot from 2024-02-14 09-38-44.png'     'Screenshot from 2024-02-21 09-59-55.png'
 git            'Screenshot from 2024-02-14 09-41-17.png'     'Screenshot from 2024-02-21 10-01-17.png'
 git2.txt       'Screenshot from 2024-02-14 09-41-44.png'     'Screenshot from 2024-02-21 10-02-00.png'
 git3.txt       'Screenshot from 2024-02-14 09-53-05.png'
 git.txt        'Screenshot from 2024-02-14 09-55-57.png'
```



```
student@student1:~/harshi$ git commit -m "savemaster"
[master ef10640] savemaster
 1 file changed, 2 insertions(+)
 create mode 100644 branch2.txt
student@student1:~/harshi$ git rebase master feature-branch2
Successfully rebased and updated refs/heads/feature-branch2.
```

*The –graph: flag enables you to view your git log as a graph.

*To make things things interesting, you can combine this command

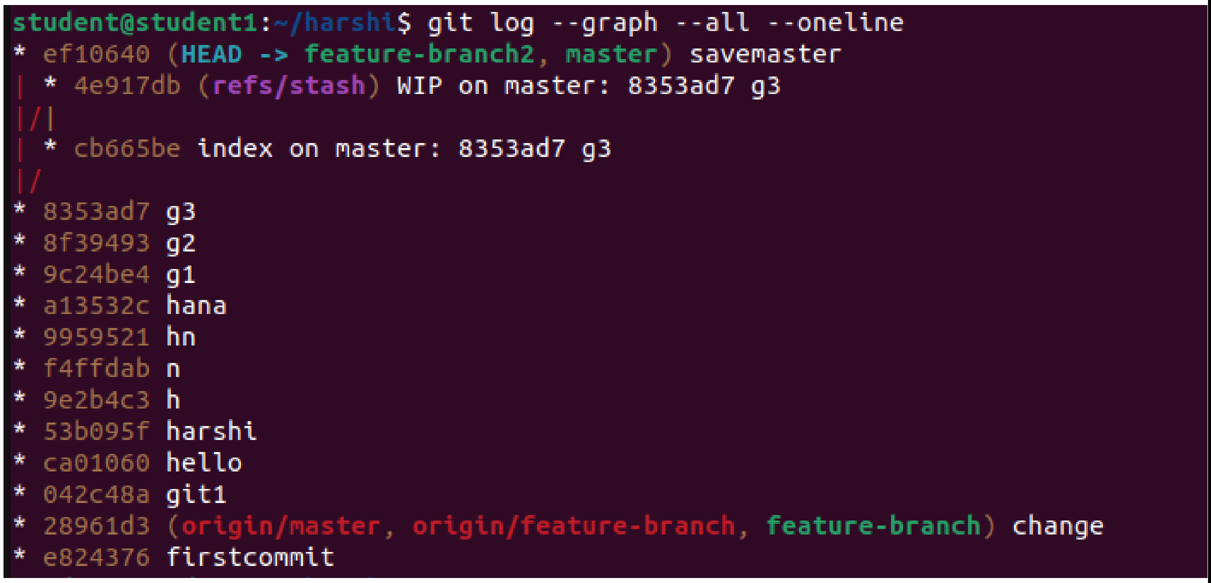
with --oneline option you learned from above.

*One of the benefit of using this command is that it enables you to get a overview of how commits have merged and how the git history was created.

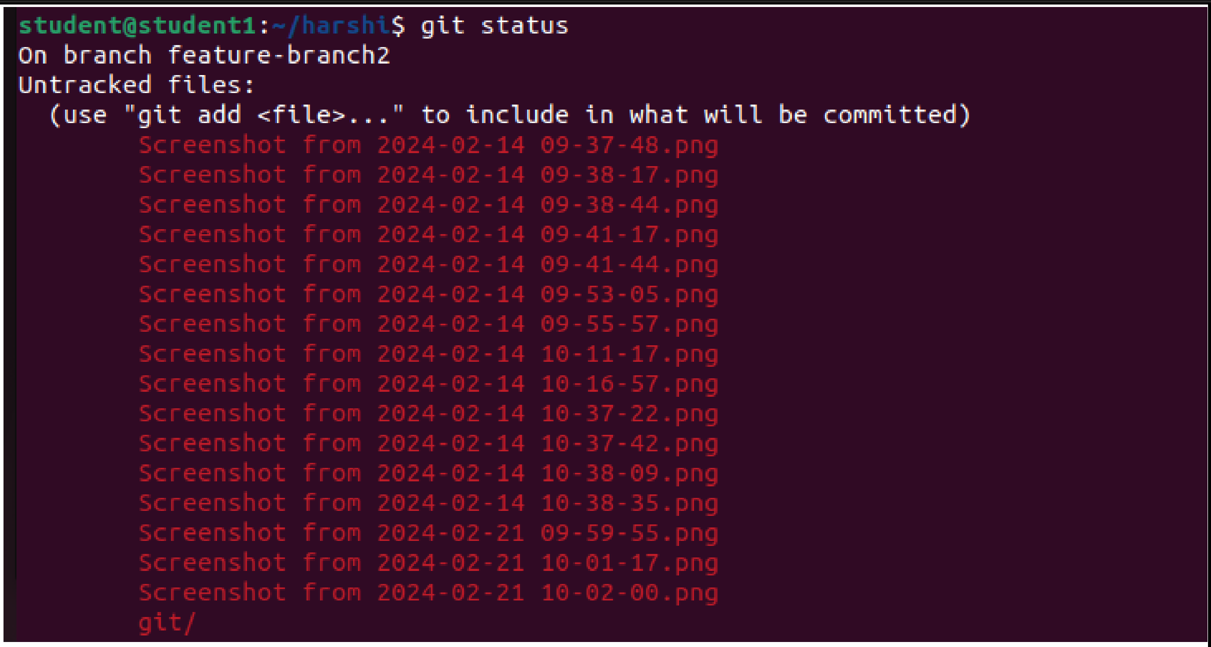
*$git log –graph –graph –all –oneline :

>A Git Graph is a pictorial representation of git commits and git actions(commands) on various branches.

> If you want to see the history of all branches/tags/etc., then you can use the --all shortcut.



```
student@student1:~/harshi$ git log --graph --all --oneline
* ef10640 (HEAD -> feature-branch2, master) savemaster
| * 4e917db (refs/stash) WIP on master: 8353ad7 g3
|/|
| * cb665be index on master: 8353ad7 g3
|/
* 8353ad7 g3
* 8f39493 g2
* 9c24be4 g1
* a13532c hana
* 9959521 hn
* f4ffdab n
* 9e2b4c3 h
* 53b095f harshi
* ca01060 hello
* 042c48a git1
* 28961d3 (origin/master, origin/feature-branch, feature-branch) change
* e824376 firstcommit
```



```
student@student1:~/harshi$ git status
On branch feature-branch2
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Screenshot from 2024-02-14 09-37-48.png
        Screenshot from 2024-02-14 09-38-17.png
        Screenshot from 2024-02-14 09-38-44.png
        Screenshot from 2024-02-14 09-41-17.png
        Screenshot from 2024-02-14 09-41-44.png
        Screenshot from 2024-02-14 09-53-05.png
        Screenshot from 2024-02-14 09-55-57.png
        Screenshot from 2024-02-14 10-11-17.png
        Screenshot from 2024-02-14 10-16-57.png
        Screenshot from 2024-02-14 10-37-22.png
        Screenshot from 2024-02-14 10-37-42.png
        Screenshot from 2024-02-14 10-38-09.png
        Screenshot from 2024-02-14 10-38-35.png
        Screenshot from 2024-02-21 09-59-55.png
        Screenshot from 2024-02-21 10-01-17.png
        Screenshot from 2024-02-21 10-02-00.png
        git/
```

EXPERIMENT NO-06

Collaboration and Remote Repositories

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

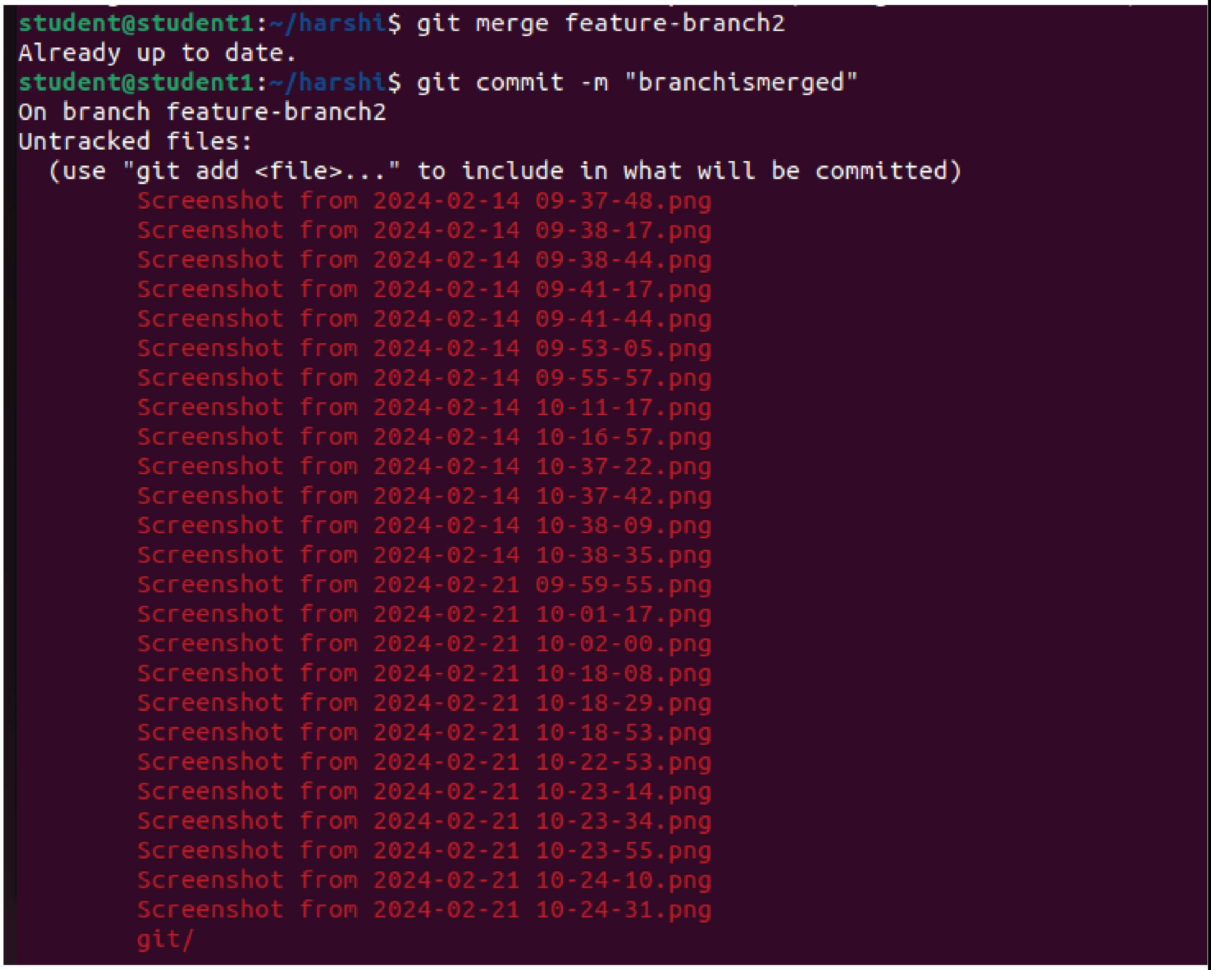
*`$ git merge feature-branch` : command is used to merge changes from the specified branch (in this case, feature-branch) into the current branch.

Typically, you execute this command while you're on the branch where you want to merge the changes. This command will incorporate the changes from featurebranch into the branch you're currently on.

After successfully merging feature-branch into master, you'll have all the changes from feature-branch incorporated into master, and you can continue working on master with the merged changes.

* `$ git commit -m “branch is merged”` : This command will commit a message that the branch is merged.

After running the git commit command, Git will create a new commit with the staged changes and associate the provided commit message with it. This helps maintain a clear history of changes in your Git repository.



```
student@student1:~/harshi$ git merge feature-branch2
Already up to date.
student@student1:~/harshi$ git commit -m "branchismerged"
On branch feature-branch2
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Screenshot from 2024-02-14 09-37-48.png
        Screenshot from 2024-02-14 09-38-17.png
        Screenshot from 2024-02-14 09-38-44.png
        Screenshot from 2024-02-14 09-41-17.png
        Screenshot from 2024-02-14 09-41-44.png
        Screenshot from 2024-02-14 09-53-05.png
        Screenshot from 2024-02-14 09-55-57.png
        Screenshot from 2024-02-14 10-11-17.png
        Screenshot from 2024-02-14 10-16-57.png
        Screenshot from 2024-02-14 10-37-22.png
        Screenshot from 2024-02-14 10-37-42.png
        Screenshot from 2024-02-14 10-38-09.png
        Screenshot from 2024-02-14 10-38-35.png
        Screenshot from 2024-02-21 09-59-55.png
        Screenshot from 2024-02-21 10-01-17.png
        Screenshot from 2024-02-21 10-02-00.png
        Screenshot from 2024-02-21 10-18-08.png
        Screenshot from 2024-02-21 10-18-29.png
        Screenshot from 2024-02-21 10-18-53.png
        Screenshot from 2024-02-21 10-22-53.png
        Screenshot from 2024-02-21 10-23-14.png
        Screenshot from 2024-02-21 10-23-34.png
        Screenshot from 2024-02-21 10-23-55.png
        Screenshot from 2024-02-21 10-24-10.png
        Screenshot from 2024-02-21 10-24-31.png
        git/
```

EXPERIMENT NO -07

Git tags and releases:

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

*Tags are reference to a specific point git history.

* Tagging is generally used to capture a point in history that is used for a version release.

*Tagging can be associated with the message.

* Using show command ,we can list out git tag names.

*$ git tag : if you run the git tag command without any arguments, it will list all the tags in your Git repository. This command is useful for viewing the existing tags in your repository.

Tags provide a way to mark specific commits in your repository's history, making it easier to reference them later. They're commonly used to mark releases, so you can easily find the commit associated with a particular version of your software.

*$ git tag v1.0 : used to create a lightweight tag in your Git repository. Tags are used to mark specific points in history, such as releases or significant milestones.

After running this command, the tag v1.0 will be created at the current commit. This tag can then be used as a reference point in your repository's history.

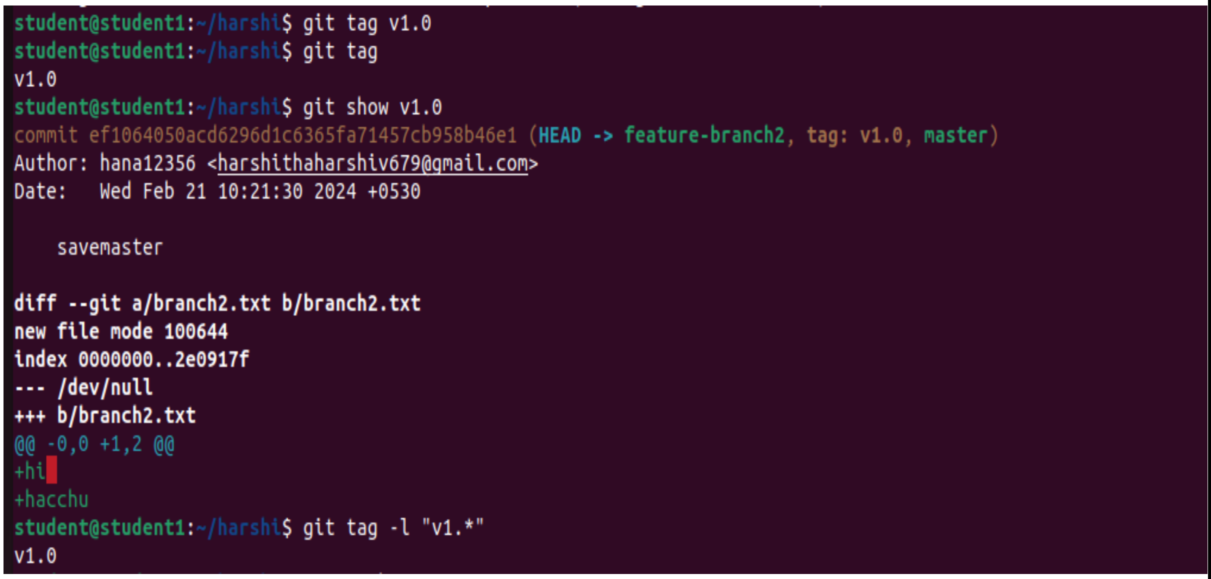
*$ git tag –a v1.1 –m "tag to release" : This command creates an annotated tag named v1.1 with the message "tag to release". Annotated tags include additional metadata such as the tagger's name, email, and the date the tag was created. The message provides additional context or information about the tag.After running this command, the tag v1.1 will be created at the current commit, and you can use it as a reference point in your repository's history.

* $ git show v1.0 : The git show command is used to display information about commits, tags, or other objects in your Git repository.

When you run git show followed by a tag name, it will display information about the specified tag. When you run this command, Git will display detailed information about the tag v1.0, including the commit it points to, the tagger information (if it's an annotated tag), and the commit message associated with the tagged commit.

If **v1.0** is an annotated tag, the output will also include any additional metadata and the tag message. If it's a lightweight tag, the output will be similar to **git show** for a commit.

This command is useful for reviewing the details of a specific tag in your repository, such as when it was created and what changes it represents.



```
student@student1:~/harshi$ git tag v1.0
student@student1:~/harshi$ git tag
v1.0
student@student1:~/harshi$ git show v1.0
commit ef1064050acd6296d1c6365fa71457cb958b46e1 (HEAD -> feature-branch2, tag: v1.0, master)
Author: hana12356 <harshithaharshiv679@gmail.com>
Date:   Wed Feb 21 10:21:30 2024 +0530

    savemaster

diff --git a/branch2.txt b/branch2.txt
new file mode 100644
index 0000000..2e0917f
--- /dev/null
+++ b/branch2.txt
@@ -0,0 +1,2 @@
+hi
+hacchu
student@student1:~/harshi$ git tag -l "v1.*"
v1.0
```

$ git tag –l "v1." : command is used to list all tags that match the specified pattern.

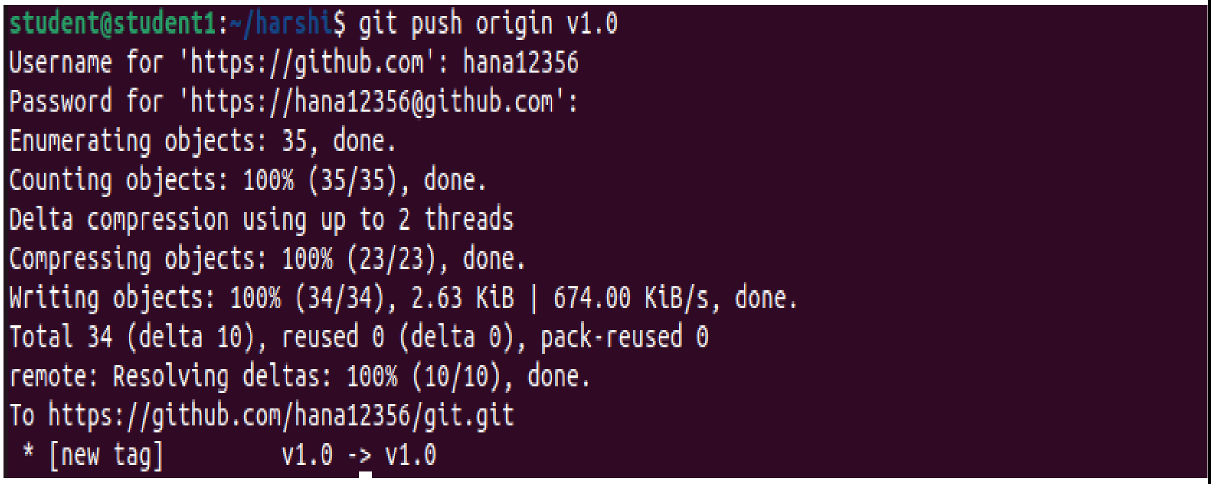
In this case, the pattern "v1.*" is a regular expression pattern that matches tags starting with

v1. followed by any characters (represented by *).

When you run this command, Git will list all tags in your repository that match the pattern **"v1.*".** This means it will list tags like v1.0, v1.1, v1.2, etc., but not tags like v2.0 or release-v1.0.

This command is useful when you want to filter and list specific tags based on a pattern or criteria. It allows you to easily find tags that match a certain versioning pattern or naming convention in your repository

*$ git push origin v1.0: This Git command pushes the local branch named "v1.0" to the remote repository named "origin." It updates the remote repository with the changes made in the local "v1.0" branch.



```
student@student1:~/harshi$ git push origin v1.0
Username for 'https://github.com': hana12356
Password for 'https://hana12356@github.com':
Enumerating objects: 35, done.
Counting objects: 100% (35/35), done.
Delta compression using up to 2 threads
Compressing objects: 100% (23/23), done.
Writing objects: 100% (34/34), 2.63 KiB | 674.00 KiB/s, done.
Total 34 (delta 10), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (10/10), done.
To https://github.com/hana12356/git.git
 * [new tag]         v1.0 -> v1.0
```

EXPERIMENT NO -08

Advanced Git Operations

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

*cherry -pick: In Git, cherry-picking is taking a single commit from one branch and adding it as the latest commit on another branch.

*$ git init : to initialize a new git repository int the current directory. When you run

this command in a directory,Git creates a new subdirectory named '.git' that contains

all of the necessary metadata for the repository. This '.git' directory is where Git

stores information about the repository's configuration,commits,branches and more.

We can start adding the files, making commits,and managing our version controlled

project using Git.

*$ git commit : The git commit command is one of the core primary functions of Git.

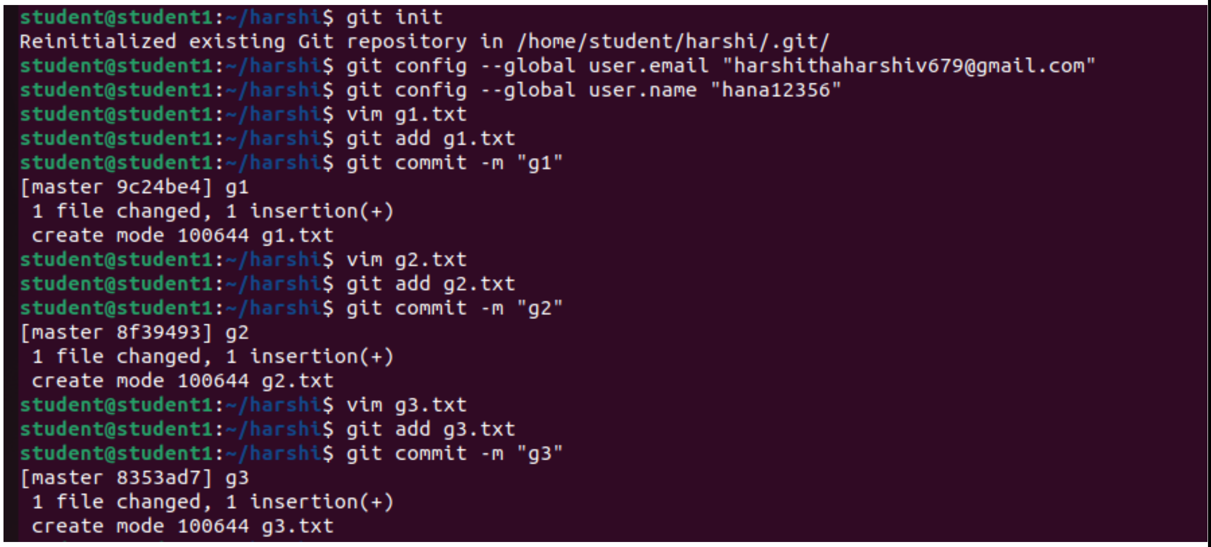
* $ vi filename.txt : this command is used to open a new file.

>create a multiple files

>then add the files using git add command

>after give commit command to save the changes for a particular file.

*$ git status : check the status whether the rebasing has done or not.

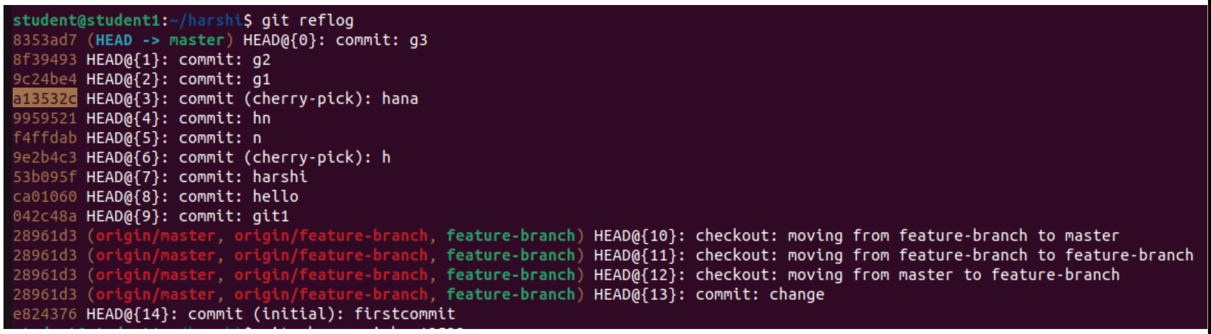


```
student@student1:~/harshi$ git init
Reinitialized existing Git repository in /home/student/harshi/.git/
student@student1:~/harshi$ git config --global user.email "harshithaharshiv679@gmail.com"
student@student1:~/harshi$ git config --global user.name "hana12356"
student@student1:~/harshi$ vim g1.txt
student@student1:~/harshi$ git add g1.txt
student@student1:~/harshi$ git commit -m "g1"
[master 9c24be4] g1
 1 file changed, 1 insertion(+)
 create mode 100644 g1.txt
student@student1:~/harshi$ vim g2.txt
student@student1:~/harshi$ git add g2.txt
student@student1:~/harshi$ git commit -m "g2"
[master 8f39493] g2
 1 file changed, 1 insertion(+)
 create mode 100644 g2.txt
student@student1:~/harshi$ vim g3.txt
student@student1:~/harshi$ git add g3.txt
student@student1:~/harshi$ git commit -m "g3"
[master 8353ad7] g3
 1 file changed, 1 insertion(+)
 create mode 100644 g3.txt
```

*$ git log: Git log is a utility tool to review and read a history of everything that happens to a repository. Multiple options can be used with a git log to make history more specific. A

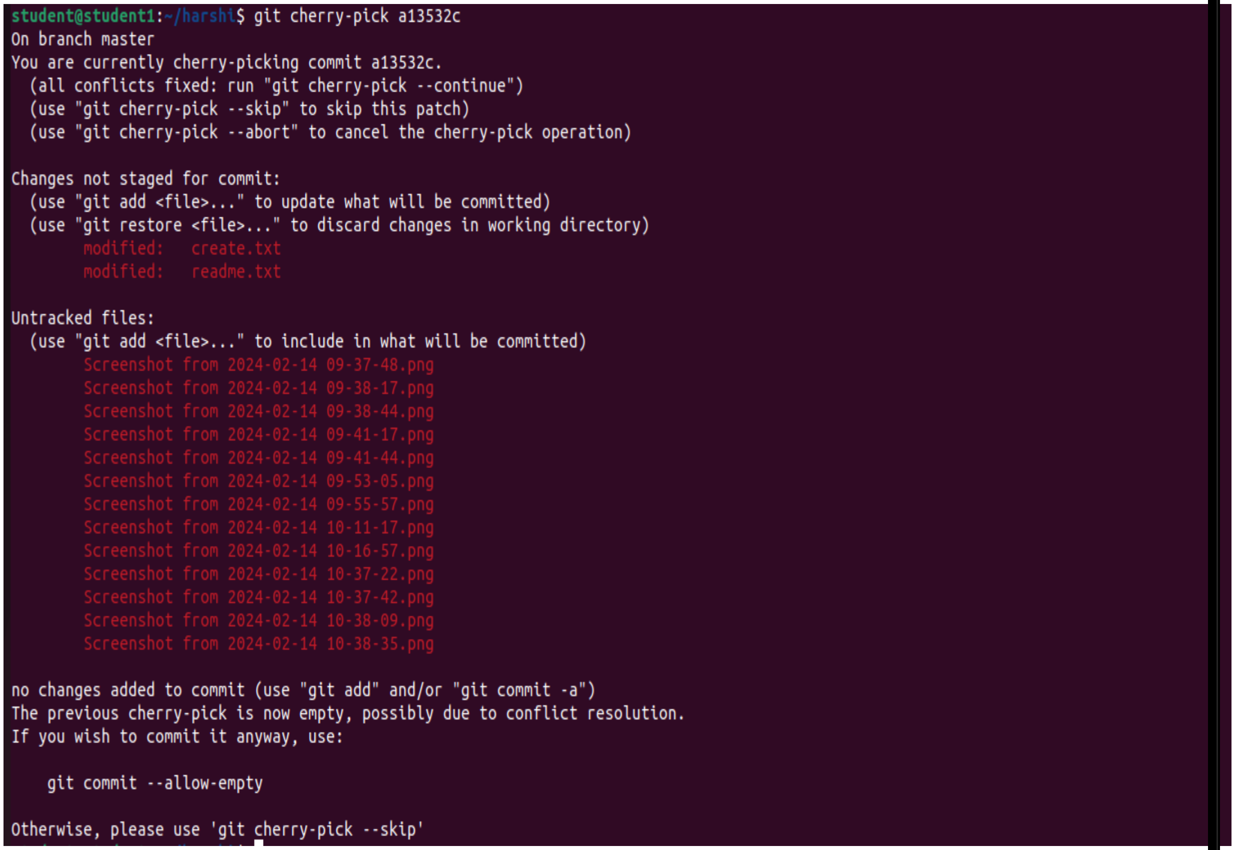
commit hash, which is a 40 character checksum data generated by SHA (Secure Hash Algorithm) algorithm. It is a unique number.

*$ git reflog: Git reflog is an isolated store used to maintain an accurate running history of modifications made to your repository's HEAD pointer.



```
student@student1:~/harshi$ git reflog
8353ad7 (HEAD -> master) HEAD@{0}: commit: g3
8f39493 HEAD@{1}: commit: g2
9c24be4 HEAD@{2}: commit: g1
a13532c HEAD@{3}: commit (cherry-pick): hana
9959521 HEAD@{4}: commit: hn
f4ffdab HEAD@{5}: commit: n
9e2b4c3 HEAD@{6}: commit (cherry-pick): h
53b095f HEAD@{7}: commit: harshi
ca01060 HEAD@{8}: commit: hello
042c48a HEAD@{9}: commit: git1
28961d3 (origin/master, origin/feature-branch, feature-branch) HEAD@{10}: checkout: moving from feature-branch to master
28961d3 (origin/master, origin/feature-branch, feature-branch) HEAD@{11}: checkout: moving from feature-branch to feature-branch
28961d3 (origin/master, origin/feature-branch, feature-branch) HEAD@{12}: checkout: moving from master to feature-branch
28961d3 (origin/master, origin/feature-branch, feature-branch) HEAD@{13}: commit: change
e824376 HEAD@{14}: commit (initial): firstcommit
```

*$ git cherry-pick commit id : Each commit has a unique hash, you need to find this hash from either the commit history on GitHub or Bitbucket or by running the 'git log --oneline' command in your terminal. Once you have the hash for the commit you want to cherry-pick, you can execute the 'git cherry-pick' command followed by the hash.



```
student@student1:~/harshi$ git cherry-pick a13532c
On branch master
You are currently cherry-picking commit a13532c.
  (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)

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Screenshot from 2024-02-14 09-37-48.png
        Screenshot from 2024-02-14 09-38-17.png
        Screenshot from 2024-02-14 09-38-44.png
        Screenshot from 2024-02-14 09-41-17.png
        Screenshot from 2024-02-14 09-41-44.png
        Screenshot from 2024-02-14 09-53-05.png
        Screenshot from 2024-02-14 09-55-57.png
        Screenshot from 2024-02-14 10-11-17.png
        Screenshot from 2024-02-14 10-16-57.png
        Screenshot from 2024-02-14 10-37-22.png
        Screenshot from 2024-02-14 10-37-42.png
        Screenshot from 2024-02-14 10-38-09.png
        Screenshot from 2024-02-14 10-38-35.png

no changes added to commit (use "git add" and/or "git commit -a")
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'
```

EXPERIMENT NO – 09

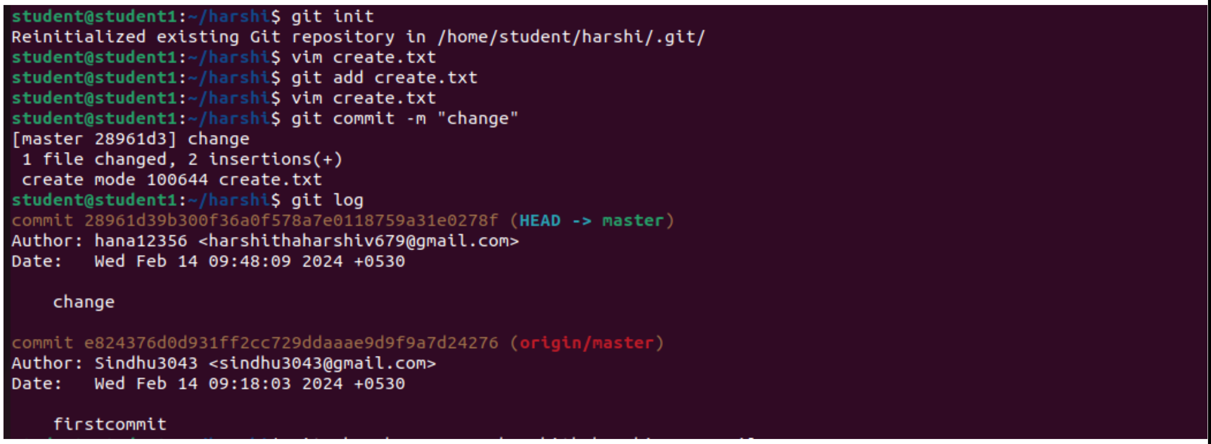
Analysing and Changing Git History

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

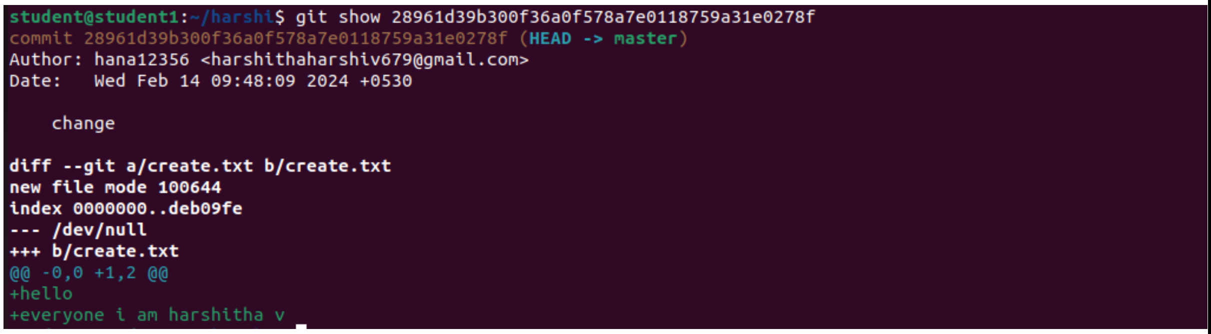
*$ git log : The git log command is used to display the commit history of the current branch in your Git repository. By default, it shows the commits starting from the most recent one and goes backward.When you run this command, Git will display a list of commits in your repository, showing information such as the commit hash, author, date, and commit message for each commit.

*$ git show : git-show is a command line utility that is used to view expanded details on Git objects such as blobs, trees, tags, and commits.

*$ git show "commit id" : To show detailed information about a specific commit identified by its commit ID (or hash), you would use the git show command followed by the commit ID. Replace <commit_id> with the actual commit ID you want to display information about.This command will display detailed information about the commit with the specified commit ID, including the commit message, author, date, and the changes introduced by the commit.



```
student@student1:~/harshi$ git init
Reinitialized existing Git repository in /home/student/harshi/.git/
student@student1:~/harshi$ vim create.txt
student@student1:~/harshi$ git add create.txt
student@student1:~/harshi$ vim create.txt
student@student1:~/harshi$ git commit -m "change"
[master 28961d3] change
 1 file changed, 2 insertions(+)
 create mode 100644 create.txt
student@student1:~/harshi$ git log
commit 28961d39b300f36a0f578a7e0118759a31e0278f (HEAD -> master)
Author: hana12356 <harshithaharshiv679@gmail.com>
Date:   Wed Feb 14 09:48:09 2024 +0530

    change

commit e824376d0d931ff2cc729ddaaae9d9f9a7d24276 (origin/master)
Author: Sindhu3043 <sindhu3043@gmail.com>
Date:   Wed Feb 14 09:18:03 2024 +0530

    firstcommit
```



```
student@student1:~/harshi$ git show 28961d39b300f36a0f578a7e0118759a31e0278f
commit 28961d39b300f36a0f578a7e0118759a31e0278f (HEAD -> master)
Author: hana12356 <harshithaharshiv679@gmail.com>
Date:   Wed Feb 14 09:48:09 2024 +0530

    change

diff --git a/create.txt b/create.txt
new file mode 100644
index 0000000..deb09fe
--- /dev/null
+++ b/create.txt
@@ -0,0 +1,2 @@
+hello
+everyone i am harshitha v
```

EXPERIMENT NO-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."

*$ git log --author= “name” --after = “yyyy-mm-dd” --before = “yyyy-mm-dd”:

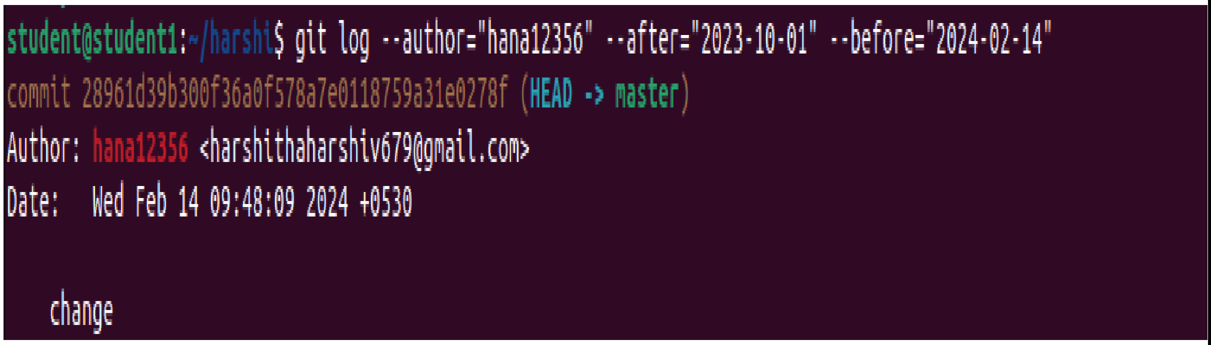
To filter the commit log by author and date range using the git log command, you can combine the --author, --after, and --before options.

*Replace "name" with the author's name, "yyyy-mm-dd" with the desired dates, and adjust the date format accordingly. Remember to enclose the author's name in quotes if it contains

spaces or special characters.

*If you want to search for commits by multiple authors, you can use --author multiple times,

or you can use a regular expression to match authors' names.



```
student@student1:~/harshi$ git log --author="hana12356" --after="2023-10-01" --before="2024-02-14"
commit 28961d39b300f36a0f578a7e0118759a31e0278f (HEAD -> master)
Author: hana12356 <harshithaharshiv679@gmail.com>
Date:   Wed Feb 14 09:48:09 2024 +0530

    change
```

EXPERIMENT NO -11

Analysing and changing Git History

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

*$ git reflog : Git uses the git reflog command to record changes made to the tips of branches.

* $ git log -n 5 : This command is used to display the last 5 commits in your

repository's commit history.

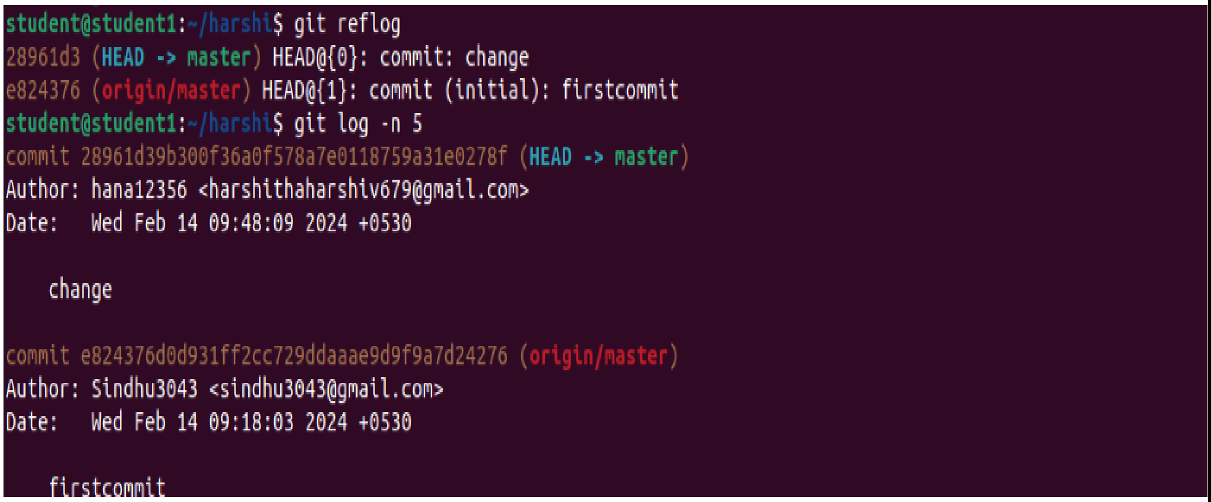
It limits the output to the specified number of commits, in this case, 5. When you run

this command, Git will display the information for the last 5 commits in your

repository, starting from the most recent commit and going backward in time.

This command is useful when you want to quickly view the most recent commits in

your repository, especially if you're only interested in a specific number of commits.



```
student@student1:~/harshi$ git reflog
28961d3 (HEAD -> master) HEAD@{0}: commit: change
e824376 (origin/master) HEAD@{1}: commit (initial): firstcommit
student@student1:~/harshi$ git log -n 5
commit 28961d39b300f36a0f578a7e0118759a31e0278f (HEAD -> master)
Author: hana12356 <harshithaharshiv679@gmail.com>
Date:   Wed Feb 14 09:48:09 2024 +0530

    change

commit e824376d0d931ff2cc729ddaaae9d9f9a7d24276 (origin/master)
Author: Sindhu3043 <sindhu3043@gmail.com>
Date:   Wed Feb 14 09:18:03 2024 +0530

    firstcommit
```

EXPERIMENT NO-12

Analysing and Changing Git History

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

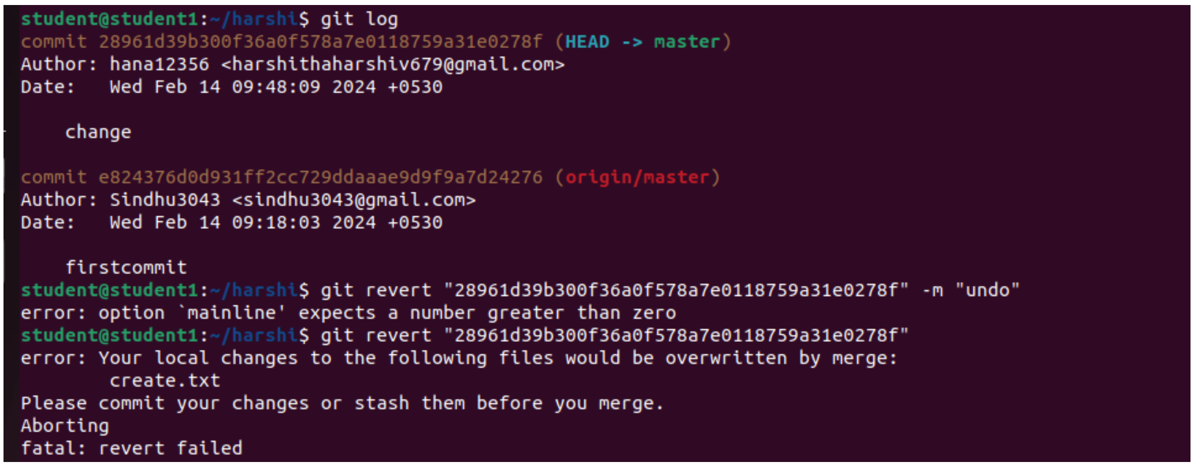
*$ git revert: The git revert command is a forward-moving undo operation that offers a safe method of undoing changes.

$ git revert "commit id" -m "revert done" : The git revert command is used to create a new commit that undoes the changes made by a specific commit or range of commits.*However, the -m option you've provided is used to specify the mainline parent number when reverting a merge commit, which isn't applicable when reverting a regular commit.

*Replace <commit_id> with the commit ID of the commit you want to revert.

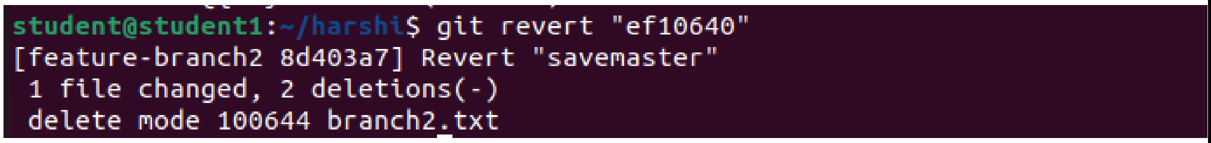
However, it's important to note that -m is used for merge commits and doesn't apply to regular commits.*After running git revert, Git will create a new commit that contains the changes to undo the specified commit. This approach allows you to keep a clean history while reverting changes in a controlled manner.

*when error occus



```
student@student1:~/harshi$ git log
commit 28961d39b300f36a0f578a7e0118759a31e0278f (HEAD -> master)
Author: hana12356 <harshithaharshiv679@gmail.com>
Date:   Wed Feb 14 09:48:09 2024 +0530

    change

commit e824376d0d931ff2cc729ddaaae9d9f9a7d24276 (origin/master)
Author: Sindhu3043 <sindhu3043@gmail.com>
Date:   Wed Feb 14 09:18:03 2024 +0530

    firstcommit
student@student1:~/harshi$ git revert "28961d39b300f36a0f578a7e0118759a31e0278f" -m "undo"
error: option `mainline' expects a number greater than zero
student@student1:~/harshi$ git revert "28961d39b300f36a0f578a7e0118759a31e0278f"
error: Your local changes to the following files would be overwritten by merge:
        create.txt
Please commit your changes or stash them before you merge.
Aborting
fatal: revert failed
```

*revert successful:



```
student@student1:~/harshi$ git revert "ef10640"
[feature-branch2 8d403a7] Revert "savemaster"
 1 file changed, 2 deletions(-)
 delete mode 100644 branch2.txt
```