VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belagavi-560014,Karnatak

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:ANUSHA J

USN: 1SV22CS008

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

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PARTICULARS OF THE EXPERIMENTS PERFORMED
CONTENTS

EXPT NO	DATE	TITLE OF THE EXPT	CONDUCTION (20M)	VIVA (10M)	TOTAL (30M)	SIGN
1	22/11/2024					
2	29/12/2024					
3	6/12/2024					
4	20/12/2024					
5	27/12/2024					
6	10/1/24					
7	10/1/24					
8	17/1/24					
9	31/1/24					
10	14/2/24					
11	21/2/24					
12	28/2/24					
		AVERAGE				

SIGNATURE OF COURSE INSTRUCTOR

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)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that, Git Lab Programs has been successfully carried out by NAME [USN] 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……	

EXPERIMENT-01

Setting up and basic commands:

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

The commands that are used here are:

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

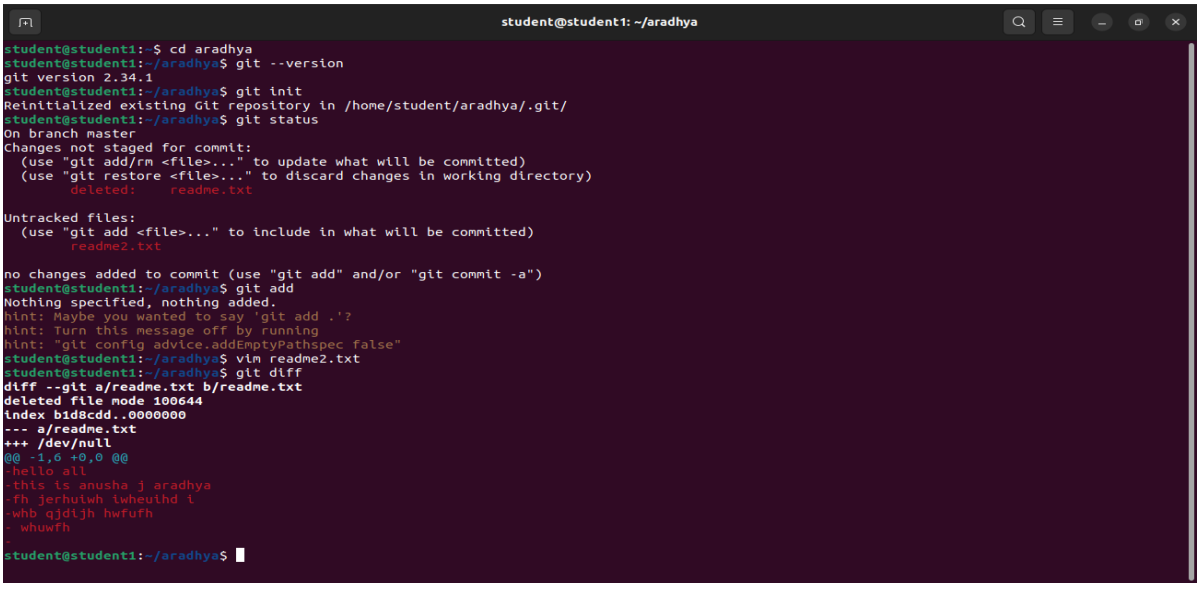
• **$ cd :** Desktop – cd(change directory),this command is used to change the present Directory into Desktop

• **$ mkdir gitlab1** : mkdir(make directory),here new directory is created which is named as gitlab1.

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

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

The above commands which are executed is shown below:



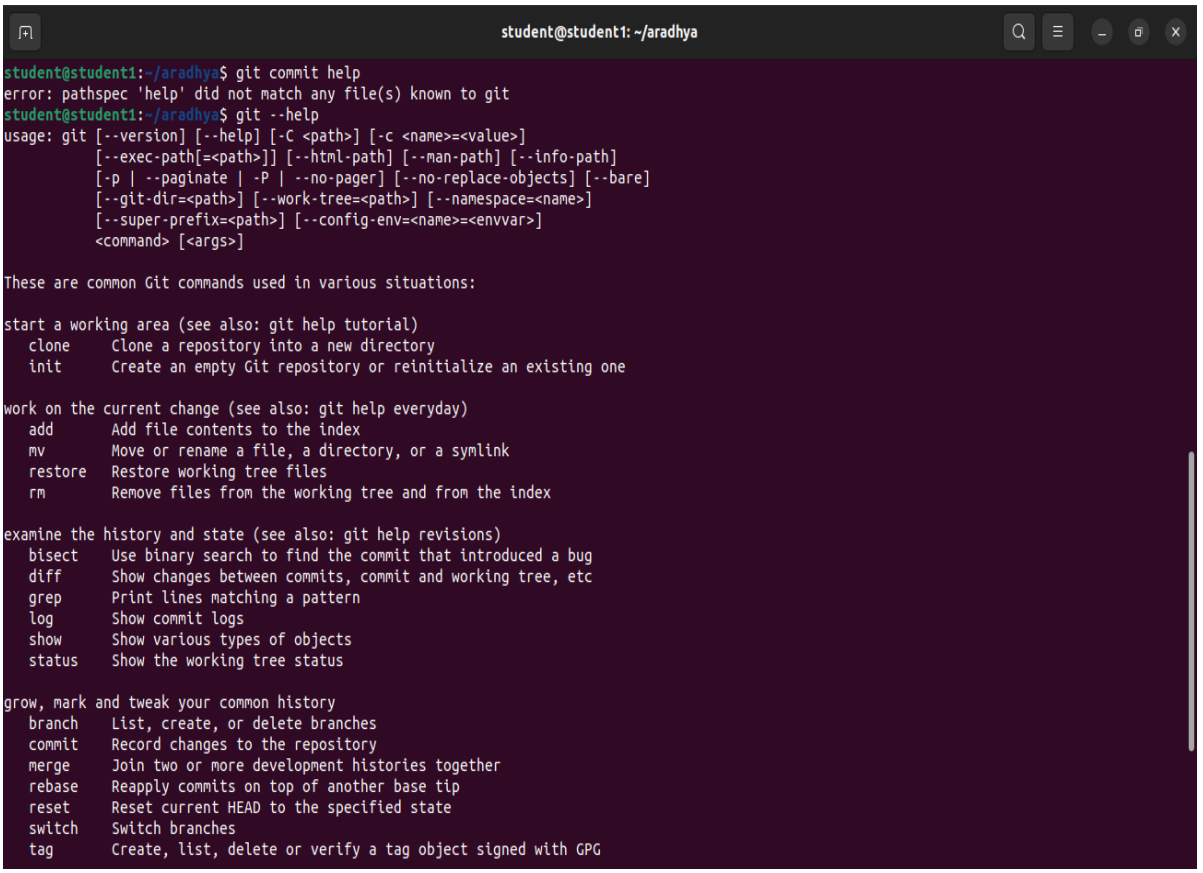
• **$ git init** : to initialize a new git repository int the current directory. When you run this command in a directory,Git creates a new subdirectory named '.

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 Gi

• **$ 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: ~/aradhya
student@student1:~/aradhya$ git commit help
error: pathspec 'help' did not match any file(s) known to git
student@student1:~/aradhya$ 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
   tag       Create, list, delete or verify a tag object signed with GPG
```

• **$ 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:~$ cd aradhya
student@student1:~/aradhya$ git --version
git version 2.34.1
student@student1:~/aradhya$ git init
Reinitialized existing Git repository in /home/student/aradhya/.git/
student@student1:~/aradhya$ git status
On branch master
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:    readme.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        readme2.txt

no changes added to commit (use "git add" and/or "git commit -a")
student@student1:~/aradhya$ git add
Nothing specified, nothing added.
```

• **$ 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 "aradhya"

• **$ git config --global user.email "your email@example.com":**command is 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" 1sv22cs008@gmail.com"

• **$ git remote add origin "remote repository URL":** ur 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:~$ cd aradhya
student@student1:~/aradhya$ git config --global user.email "1sv22cs008@gmail.com"
student@student1:~/aradhya$ git config --global user.name "anusha-j-aradhya"
student@student1:~/aradhya$ git remote add origin "https://github.com/anusha-j-aradhya/gitlab.git"
student@student1:~/aradhya$ git remote add origin "https://github.com/anusha-j-aradhya/gitlab.git"
error: remote origin already exists.
student@student1:~/aradhya$ git push origin master
Username for 'https://github.com': anusha-j-aradhya
Password for 'https://anusha-j-aradhya@github.com':
[1]+  Stopped                 git push origin master
student@student1:~/aradhya$ git push origin master
Username for 'https://github.com': "anusha-j-aradhya"
Password for 'https://"anusha-j-aradhya"@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for infor
mation on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/anusha-j-aradhya/gitlab.git/'
student@student1:~/aradhya$ git push origin master^C
student@student1:~/aradhya$ ^C
student@student1:~/aradhya$ git push origin master
Username for 'https://github.com': anusha-j-aradhya
Password for 'https://anusha-j-aradhya@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for infor
mation on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/anusha-j-aradhya/gitlab.git/'
student@student1:~/aradhya$ git remote -v
origin  https://github.com/anusha-j-aradhya/gitlab.git (fetch)
origin  https://github.com/anusha-j-aradhya/gitlab.git (push)
```

¬ 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.

EXPERIMENT-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 feature branch 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

```
  (use "git restore --staged <file>..." to unstage)
        new file:   branchfile2.txt
        new file:   branchfile3.txt
        new file:   branchfile4.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        branchfile.txt
        gitlab/
        readme2.txt

student@student1:~/aradhya$ git checkout feature_branch2
A       branchfile2.txt
A       branchfile3.txt
A       branchfile4.txt
Switched to branch 'feature_branch2'
student@student1:~/aradhya$ vi branchfile6.txt
student@student1:~/aradhya$ git add branchfile6.txt
student@student1:~/aradhya$ git staus
git: 'staus' is not a git command. See 'git --help'.

The most similar command is
        status
student@student1:~/aradhya$ git status
On branch feature_branch2
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branchfile2.txt
        new file:   branchfile3.txt
        new file:   branchfile4.txt
        new file:   branchfile6.txt

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

```
student@student1: ~/aradhya
        readme2.txt

student@student1:~/aradhya$ git checkout mater
error: pathspec 'mater' did not match any file(s) known to git
student@student1:~/aradhya$ git checkout master
A       branchfile2.txt
A       branchfile3.txt
A       branchfile4.txt
A       branchfile6.txt
Switched to branch 'master'
student@student1:~/aradhya$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branchfile2.txt
        new file:   branchfile3.txt
        new file:   branchfile4.txt
        new file:   branchfile6.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        branchfile.txt
        gitlab/
        readme2.txt

student@student1:~/aradhya$ ls
branchfile2.txt  branchfile3.txt  branchfile4.txt  branchfile6.txt  branchfile.txt  gitlab  readme2.txt  readme.txt
student@student1:~/aradhya$ git commit -m "integrated file"
[master 905df36] integrated file
 4 files changed, 9 insertions(+)
 create mode 100644 branchfile2.txt
 create mode 100644 branchfile3.txt
 create mode 100644 branchfile4.txt
 create mode 100644 branchfile6.txt
student@student1:~/aradhya$ git rebase master feature branch2
```

• **$ git push origin feature-branch :** used to push the commits from your local feature branch 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

```
student@student1:~/aradhya$ git checkout mater
error: pathspec 'mater' did not match any file(s) known to git
student@student1:~/aradhya$ git checkout master
A       branchfile2.txt
A       branchfile3.txt
A       branchfile4.txt
A       branchfile6.txt
Switched to branch 'master'
student@student1:~/aradhya$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branchfile2.txt
        new file:   branchfile3.txt
        new file:   branchfile4.txt
        new file:   branchfile6.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        branchfile.txt
        gitlab/
        readme2.txt

student@student1:~/aradhya$ ls
branchfile2.txt  branchfile3.txt  branchfile4.txt  branchfile6.txt  branchfile.txt  gitlab  readme2.txt  readme.txt
student@student1:~/aradhya$ git commit -m "integrated file"
[master 905df36] integrated file
 4 files changed, 9 insertions(+)
 create mode 100644 branchfile2.txt
 create mode 100644 branchfile3.txt
```

EXPERIMENT-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 stack.

```
student@student1:~/aradhya$
student@student1:~/aradhya$ git clone  "https://github.com/anusha-j-aradhya/gitlab.git"
fatal: destination path 'gitlab' already exists and is not an empty directory.
student@student1:~/aradhya$ git stash pop
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branchfile2.txt
        new file:   branchfile3.txt
        new file:   branchfile4.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        branchfile.txt
        gitlab/
        readme2.txt

Dropped refs/stash@{0} (afd635f5e5002bef574f3d5dcf35df679f9a9d9a)
student@student1:~/aradhya$ git stash list
stash@{0}: WIP on feature-branch: 4b2842f hii all
student@student1:~/aradhya$
```

EXPERIMENT-04

$ 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.

```
Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git add anu.txt
warning: in the working copy of 'anu.txt', LF will be replaced by CRLF the next time Git touches it

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git commit -m"nn"
[master 8072122] nn
 1 file changed, 4 insertions(+)
 create mode 100644 anu.txt

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git clone"https://github.com/anusha-j-aradhya/gitlab"
git: 'clonehttps://github.com/anusha-j-aradhya/gitlab' is not a git command. See 'git --help'.

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ |
```

EXPERIMENT-05

Collaborate 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.

• **$ 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.

•

```
                          automatically re-schedule any  exec  that fails
    --reapply-cherry-picks
                          apply all changes, even those already present upstream

student@student1:~/aradhya$ git rebase master feature_branch2
Successfully rebased and updated refs/heads/feature_branch2.
student@student1:~/aradhya$ git log -graph all -oneline
fatal: ambiguous argument 'all': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'
student@student1:~/aradhya$ git status
On branch feature_branch2
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        branchfile.txt
        gitlab/
        readme2.txt

nothing added to commit but untracked files present (use "git add" to track)
student@student1:~/aradhya$ git merge feature_branch2
Already up to date.
student@student1:~/aradhya$ git commit -"combine"
fatal: could not lookup commit ombine
student@student1:~/aradhya$ git commit -m"combine"
On branch feature_branch2
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        branchfile.txt
        gitlab/
        readme2.txt

nothing added to commit but untracked files present (use "git add" to track)
student@student1:~/aradhya$ git checkout feature_branch2
```

check the status whether the rebasing has done or not.

EXPERIMENT-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 feature branch 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.

```
  (use "git add <file>..." to include in what will be committed)
        branchfile.txt
        gitlab/
        readme2.txt

nothing added to commit but untracked files present (use "git add" to track)
student@student1:~/aradhya$ git merge feature_branch2
Already up to date.
student@student1:~/aradhya$ git commit -"combine"
fatal: could not lookup commit ombine
student@student1:~/aradhya$ git commit -m"combine"
On branch feature_branch2
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        branchfile.txt
        gitlab/
        readme2.txt

nothing added to commit but untracked files present (use "git add" to track)
student@student1:~/aradhya$ git checkout feature_branch2
```

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

EXPERMINT-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 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 :** 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 –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.

- $ 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.

```
Reinitialized existing Git repository in /home/student/aradhya/.git/
student@student1:~/aradhya$ git tag v3.0
student@student1:~/aradhya$ git tag
v1.0
v1.1
v3.0
student@student1:~/aradhya$ git tag -a v3.1 -m"tag release"
student@student1:~/aradhya$ git tag
v1.0
v1.1
v3.0
v3.1
student@student1:~/aradhya$ git show v3.0
commit 439fb47a316f36cbf875a31947e4a4336a00f281 (HEAD -> feature-branch2, tag: v3.1, tag: v3.0, tag: v1.1, tag: v1.0)
Author: Divyashree2005 <divyashreetd2005@gmail.com>
Date:   Wed Feb 21 09:40:58 2024 +0530

    git3

diff --git a/readme11.txt b/readme11.txt
new file mode 100644
index 0000000..17ddd03
--- /dev/null
+++ b/readme11.txt
@@ -0,0 +1,4 @@
+
+what is benifit of git??
+
+GitHub allows you to create, store, change, merge, and collaborate on files or code. Any member of a team can access the GitHub repository (think of
this as a folder for files) and see the most recent version in real-time. Then, they can make edits or changes that the other collaborators also see.2
9
```

It allows you to easily find tags that match a certain versioning pattern or naming convention in your repository.

• **$ 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.

```
student@student1:~/aradhya$ git show v3.0
commit 439fb47a316f36cbf875a31947e4a4336a00f281 (HEAD -> feature-branch2, tag: v3.1, tag: v3.0, tag: v1.1, tag: v1.0)
Author: Divyashree2005 <divyashreetd2005@gmail.com>
Date:   Wed Feb 21 09:40:58 2024 +0530

    git3

diff --git a/readme11.txt b/readme11.txt
new file mode 100644
index 0000000..17ddd03
--- /dev/null
+++ b/readme11.txt
@@ -0,0 +1,4 @@
+
+what is benifit of git??
+
+GitHub allows you to create, store, change, merge, and collaborate on files or code. Any member of a team can access the GitHub repository (think of
this as a folder for files) and see the most recent version in real-time. Then, they can make edits or changes that the other collaborators also see.2
9
student@student1:~/aradhya$ git tag -l "v3.0*"
v3.0
student@student1:~/aradhya$ git push origin v3.0
Username for 'https://github.com': anusha-j-aradhya
Password for 'https://anusha-j-aradhya@github.com':
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 2 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (17/17), 2.27 KiB | 1.14 MiB/s, done.
Total 17 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), completed with 1 local object.
To https://github.com/anusha-j-aradhya/gitlab.git
 * [new tag]         v3.0 -> v3.0
student@student1:~/aradhya$
```

EXPERIMENT-08

Advanced Git Operations

Write the command to cherry-pick a range of commits from "source-branch" to the current branch To cherry-pick a range of commits from "source-branch" to the current branch, use the following command: git cherry-pick (commit id) For example, if you want to cherry-pick the commits with hashes abc123 to def456,

the command would be: git cherry-pick abc123..def456 This command applies the changes introduced by the specified range of commits to the current branch, effectively cherry-picking them

```
Admin@DESKTOP-77O5R3A MINGW64 ~ (master)
$ cd aradhya

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git --version
git version 2.43.0.windows.1

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git init
Reinitialized existing Git repository in C:/Users/Admin/aradhya/.git/

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git config --global user.email"1sv22cs008@gmail.com"

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git config --global user.name"anusha-j-aradhya"

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ vim 1c.txt

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ add 1c.txt
bash: add: command not found

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git add 1c.txt
warning: in the working copy of '1c.txt', LF will be replaced by CRLF the next time Git touches it

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git commit -m"ji"
[master de650f5] ji
 1 file changed, 2 insertions(+)
 create mode 100644 1c.txt

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git reflog
de650f5 (HEAD -> master) HEAD@{0}: commit: ji
af7ae56 HEAD@{1}: revert: Revert "hhh"
5421c31 HEAD@{2}: commit (cherry-pick): hhh
61a5426 HEAD@{3}: commit: hii
30a3ace HEAD@{4}: commit: dss
b71aa64 HEAD@{5}: commit: hiiii
eb310a3 HEAD@{6}: commit (initial): hii

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git cheery-pick de650f5
git: 'cheery-pick' is not a git command. See 'git --help'.
```

EXPERIMENT-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 "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 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.

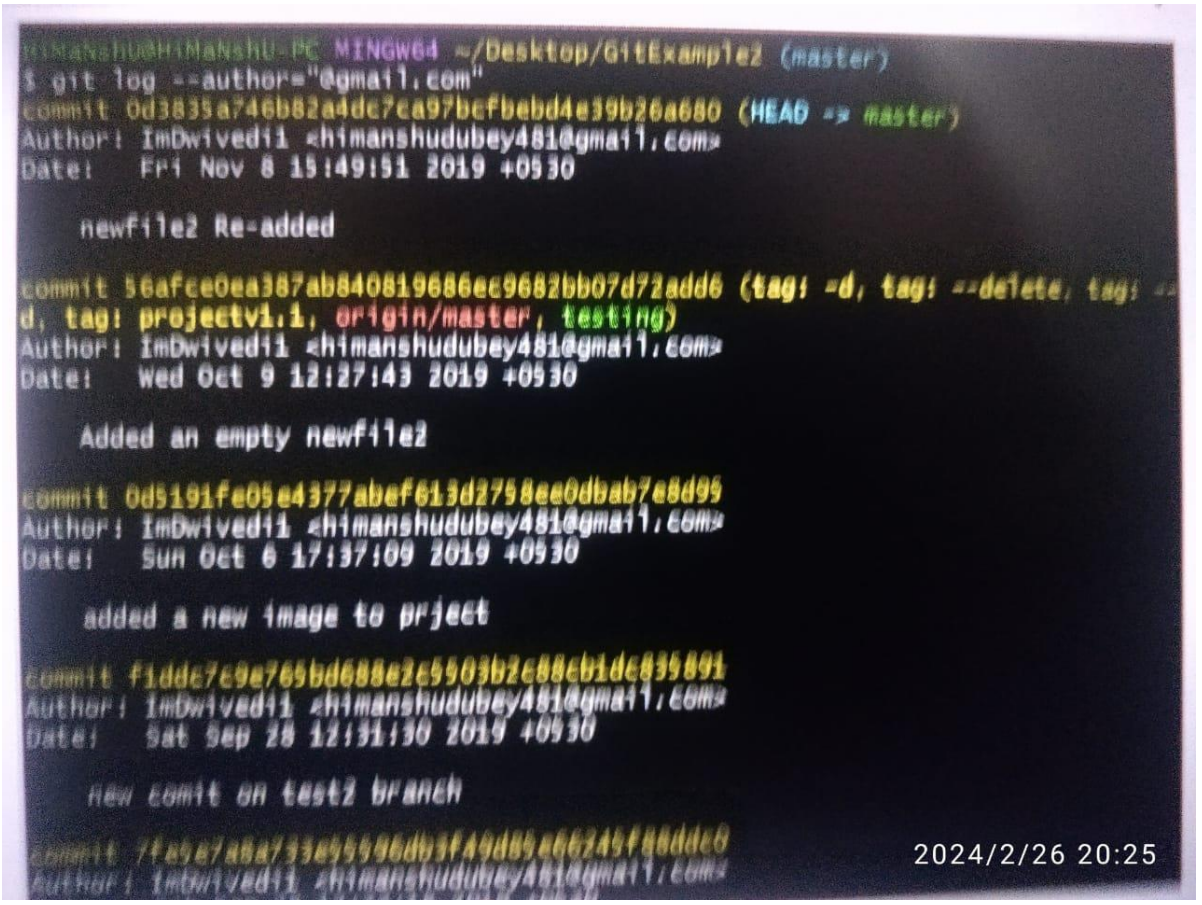
```
Admin@DESKTOP-77O5R3A MINGW64 ~ (master)
$ cd aradhya

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git --version
git version 2.43.0.windows.1

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git init
Reinitialized existing Git repository in C:/Users/Admin/aradhya/.git/

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git config --global user.email"1sv22cs008@gmail.com"

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git config --global user.name"anusha-j-aradhya"

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ vim 1c.txt

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ add 1c.txt
bash: add: command not found

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git add 1c.txt
warning: in the working copy of '1c.txt', LF will be replaced by CRLF the next time Git touches it

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git commit -m"ji"
[master de650f5] ji
 1 file changed, 2 insertions(+)
 create mode 100644 1c.txt

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git reflog
de650f5 (HEAD -> master) HEAD@{0}: commit: ji
af7ae56 HEAD@{1}: revert: Revert "hhh"
5421c31 HEAD@{2}: commit (cherry-pick): hhh
61a5426 HEAD@{3}: commit: hii
30a3ace HEAD@{4}: commit: dss
b71aa64 HEAD@{5}: commit: hiiii
eb310a3 HEAD@{6}: commit (initial): hii

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git cheery-pick de650f5
git: 'cheery-pick' is not a git command. See 'git --help'.
```

EXPERIMENT-10

Analysing and changing Git History: Write the command to list all commits made by author.

• **$ 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 containsspaces 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.



EXPERIMENT-11

Analysing and changing Git History: Write the command to display the last five commits in the repository's history.

- **$ 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.

```
Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git log -n5
commit de650f5da4d17348e9f1521ebb41b0f9eb0bd802 (HEAD -> master)
Author: anusha-j-aradhya <1sv22cs008@gamil.com>
Date:   Mon Feb 26 18:50:15 2024 +0530

    ji

commit af7ae56f142a91372c9a3d5a42faab5929ab16e4
Author: anusha-j-aradhya <1sv22cs008@gamil.com>
Date:   Mon Feb 26 15:13:16 2024 +0530

    Revert "hhh"

    This reverts commit 5421c31c9a70d3a43cf02f0a2ef1bd677298fd55.

commit 5421c31c9a70d3a43cf02f0a2ef1bd677298fd55
Author: anusha-j-aradhya <1sv22cs008@gamil.com>
Date:   Mon Feb 26 15:02:00 2024 +0530

    hhh

commit 61a54265985b53694343495bfc5394aac6ea5ebe
Author: anusha-j-aradhya <1sv22cs008@gamil.com>
Date:   Mon Feb 26 15:02:00 2024 +0530

    hii
```

EXPERIMENT-12

Analysing and changing Git History: Write the command to undo the changes introduced by the commit with the ID “abc123”.

• **$ 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 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
.

```
Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git reflog
5eb18ac (HEAD -> master) HEAD@{0}: revert: Revert "ji"
de650f5 HEAD@{1}: commit: ji
af7ae56 HEAD@{2}: revert: Revert "hhh"
5421c31 HEAD@{3}: commit (cherry-pick): hhh
61a5426 HEAD@{4}: commit: hii
30a3ace HEAD@{5}: commit: dss
b71aa64 HEAD@{6}: commit: hiiii
eb310a3 HEAD@{7}: commit (initial): hii

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
$ git revert "de650f5"
On branch master
nothing to commit, working tree clean

Admin@DESKTOP-77O5R3A MINGW64 ~/aradhya (master)
```