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: CHANDRASHEKHAR.M.PATIL
USN: 1SV22CS023

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)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that, Git Lab Programs has been successfully carried out by CHANDRASHEKHAR.M.PATIL[1SV22CS023]in partial fulfillment for 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. BASAVESHAD. 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 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.
- **$ 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:

```
chandru@dell MINGW64 ~/chandru
$ ls
githublab/

chandru@dell MINGW64 ~/chandru
$ cd githublab

chandru@dell MINGW64 ~/chandru/githublab
$ vi testfile.txt

chandru@dell MINGW64 ~/chandru/githublab
$ git --version
git version 2.43.0.windows.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.

- **$ 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.

```
dru@dell MINGW64 ~/chandru/githublab
t init
ialized empty Git repository in C:/Users/shiva/chandru/githublab/.git/

dru@dell MINGW64 ~/chandru/githublab (master)
t status
ranch master

ommits yet

acked files:
se "git add <file>..." to include in what will be committed)
    testfile.txt

ing added to commit but untracked files present (use "git add" to track)
```

- **$ 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.

```
dru@dell MINGW64 ~/chandru/githublab (master)
t help
e: git [-v | --version] [-h | --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>]
       [--config-env=<name>=<envvar>] <command> [<args>]

e are common Git commands used in various situations:

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

 on the current change (see also: git help everyday)
dd       Add file contents to the index
v        Move or rename a file, a directory, or a symlink
estore   Restore working tree files
n        Remove files from the working tree and from the index

ine the history and state (see also: git help revisions)
isect    Use binary search to find the commit that introduced a bug
iff      Show changes between commits, commit and working tree, etc
rep      Print lines matching a pattern
og       Show commit logs
how      Show various types of objects
tatus    Show the working tree status

, mark and tweak your common history
ranch    List, create, or delete branches
ommit    Record changes to the repository
erge     Join two or more development histories together
ebase    Reapply commits on top of another base tip
eset     Reset current HEAD to the specified state
witch    Switch branches
ag       Create, list, delete or verify a tag object signed with GPG

aborate (see also: git help workflows)
etch     Download objects and refs from another repository
ull      Fetch from and integrate with another repository or a local branch
ush      Update remote refs along with associated objects

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

- **$ 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.

```
chandru@dell MINGW64 ~/chandru/githublab (master)
$ git add testfile.txt

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git commit -m "add a new file"
[master (root-commit) b2b51bf] add a new file
 1 file changed, 1 insertion(+)
 create mode 100644 testfile.txt

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git status
On branch master
nothing to commit, working tree clean
```

- **$ 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 "rachanacp"

  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 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 "rachana.c.p04@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 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.

```
chandru@dell MINGW64 ~/chandru/githublab (master)
$ git config --global user.name "ChandrashekharMPatil"

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git config --global user.email "chandrashekharmpatil0@gmail.com"

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git remote add origin "https://github.com/ChandrashekharMPatil/github-report.git"

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 246 bytes | 246.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ChandrashekharMPatil/github-report.git
 * [new branch]      master -> master

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git push origin master
Everything up-to-date

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git remote -v
origin  https://github.com/ChandrashekharMPatil/github-report.git (fetch)
origin  https://github.com/ChandrashekharMPatil/github-report.git (push)
```

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

```
chandru@dell MINGW64 ~/chandru/githublab (master)
$ git status
On branch master
nothing to commit, working tree clean

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git branch feature-branch

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git status
On branch master
nothing to commit, working tree clean

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git checkout feature-branch
Switched to branch 'feature-branch'

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ vi branchfile.txt

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git add branchfile.txt
warning: in the working copy of 'branchfile.txt', LF will be replaced by CRLF the next time Git touches it

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git add branchfile.txt

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git status
On branch feature-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branchfile.txt
```

- **$ 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

```
chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git push origin feature-branch
Total 0 (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/ChandrashekharMPatil/github-report/pull/new/feature-branch
remote:
To https://github.com/ChandrashekharMPatil/github-report.git
 * [new branch]      feature-branch -> feature-branch

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git checkout master
Switched to branch 'master'
A       branchfile.txt

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branchfile.txt
```

  commits, etc.

- **$ 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.

- **$ 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.

- **$ 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.

```
chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git push origin feature-branch
Total 0 (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/ChandrashekharMPatil/github-report/pull/new/feature-branch
remote:
To https://github.com/ChandrashekharMPatil/github-report.git
 * [new branch]      feature-branch -> feature-branch

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git checkout master
Switched to branch 'master'
A       branchfile.txt

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branchfile.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.

```
chandru@dell MINGW64 ~/chandru/githublab (master)
$ git branch feature-branch
fatal: a branch named 'feature-branch' already exists

chandru@dell MINGW64 ~/chandru/githublab (master)
$ vi branchfile.txt

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git add branchfile.txt

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git stash
Saved working directory and index state WIP on master: b2b51bf add a new file

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git checkout feature-branch
Switched to branch 'feature-branch'

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git stash apply
On branch feature-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   branchfile.txt


chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git stash list
stash@{0}: WIP on master: b2b51bf add a new file
```

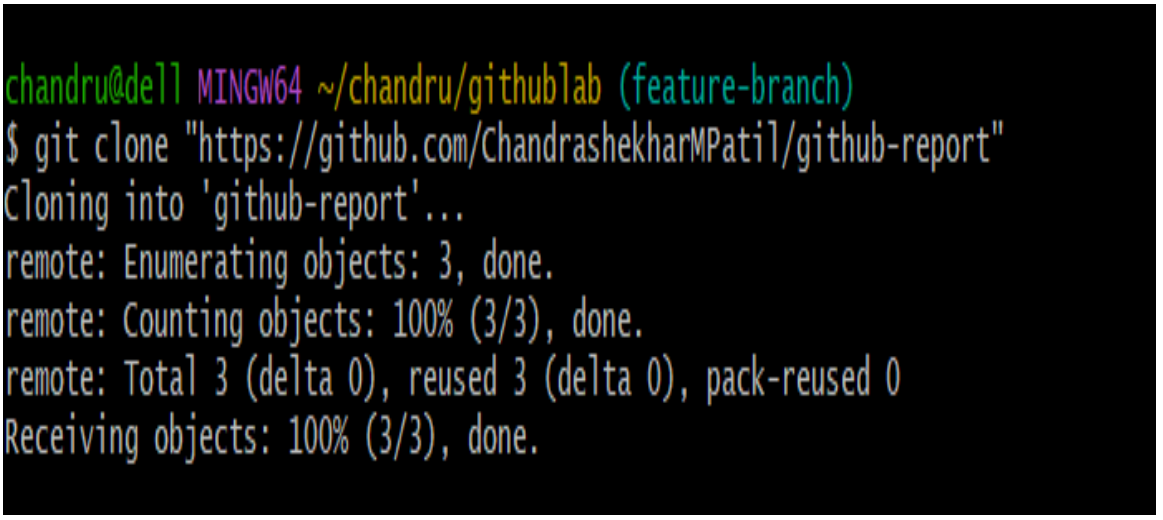
EXPERIMENT-04

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.



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.

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

```
chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git checkout master
Switched to branch 'master'
A       branchfile.txt

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git commit-m" "
git: 'commit-m ' is not a git command. See 'git --help'.

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git commit -m" "
Aborting commit due to empty commit message.

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git commit -m "rebasing"
[master 922f4b7] rebasing
 1 file changed, 1 insertion(+)
 create mode 100644 branchfile.txt

chandru@dell MINGW64 ~/chandru/githublab (master)
$ git rebase master feature-branch
Successfully rebased and updated refs/heads/feature-branch.

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git status
On branch feature-branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        github-report/

nothing added to commit but untracked files present (use "git add" to track)
```

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.

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

```
chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git merge feature-branch
Already up to date.

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git commit -m "branch is merged"
On branch feature-branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        github-report/

nothing added to commit but untracked files present (use "git add" to track)

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git log --graph --all --oneline
* 922f4b7 (HEAD -> feature-branch, master) rebasing
| * ca2fac1 (refs/stash) WIP on master: b2b51bf add a new file
|/|
| * dd98ef7 index on master: b2b51bf add a new file
|/
* b2b51bf (origin/master, origin/feature-branch) add a new file
```

EXPERIMENT -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.

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.

- **$ 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.

```
chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git tag v1.0

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git tag
v1.0

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git tag -a v1.1 -m "tag to release"

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git tag
v1.0
v1.1

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git show v1.0
commit 922f4b71bd29ea30803856da0a170c5c4e457265 (HEAD -> feature-branch, tag: v1.1, tag: v1.0, master)
Author: ChandrashekharMPatil <chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 17:58:18 2024 +0530

    rebasing

diff --git a/branchfile.txt b/branchfile.txt
new file mode 100644
index 0000000..87a43c7
--- /dev/null
+++ b/branchfile.txt
@@ -0,0 +1 @@
+hi chandru.mp

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git tag -l "v1.*"
v1.0
v1.1
```

EXPRIMENT 8

ADVANCED GIT OPERATIONS

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

•**$ git reflog :** the git reflog is a very useful tool for developers. It keep track of when the tips of branches and other references in the local repository are modified.

•**$ git cherry-pick <commit id>**: here we need to replace the <commit id > with the commit id of 7 character or references that defines the range of commits you want to cherry-pick .

•Cherry-picking is a very strong function that choose any commit in the history and implemented its feature to the amount master.

After running this command , git will apply the specified range of commits Onto your current branch .if there are any conflicts, git will pause the cherry-pick process and ask you to resolve them. After resolving conflicts , you can continue the cherry-pick with: git cherry-pick –continue .

```
chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git reflog
5cacb14 (HEAD -> feature-branch) HEAD@{0}: revert: Revert "add a new file"
922f4b7 (tag: v1.1, tag: v1.0, master) HEAD@{1}: rebase (finish): returning to refs/heads/feature-branch
922f4b7 (tag: v1.1, tag: v1.0, master) HEAD@{2}: rebase (start): checkout master
922f4b7 (tag: v1.1, tag: v1.0, master) HEAD@{3}: commit: rebasing
b2b51bf (origin/master, origin/feature-branch) HEAD@{4}: checkout: moving from feature-branch to master
b2b51bf (origin/master, origin/feature-branch) HEAD@{5}: checkout: moving from master to feature-branch
b2b51bf (origin/master, origin/feature-branch) HEAD@{6}: reset: moving to HEAD
b2b51bf (origin/master, origin/feature-branch) HEAD@{7}: checkout: moving from feature-branch to master
b2b51bf (origin/master, origin/feature-branch) HEAD@{8}: checkout: moving from master to feature-branch
b2b51bf (origin/master, origin/feature-branch) HEAD@{9}: commit (initial): add a new file

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git log
commit 5cacb14ef32f5f8188acecf104317af6beeaf37a (HEAD -> feature-branch)
Author: ChandrashekharMPatil <chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 18:14:45 2024 +0530

    Revert "add a new file"

    This reverts commit b2b51bf2b5d11da0531ea2affbb1bfe2a30f37a5.

commit 922f4b71bd29ea30803856da0a170c5c4e457265 (tag: v1.1, tag: v1.0, master)
Author: ChandrashekharMPatil <chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 17:58:18 2024 +0530

    rebasing

commit b2b51bf2b5d11da0531ea2affbb1bfe2a30f37a5 (origin/master, origin/feature-branch)
Author: ChandrashekharMPatil <Chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 17:29:33 2024 +0530

    add a new file
```

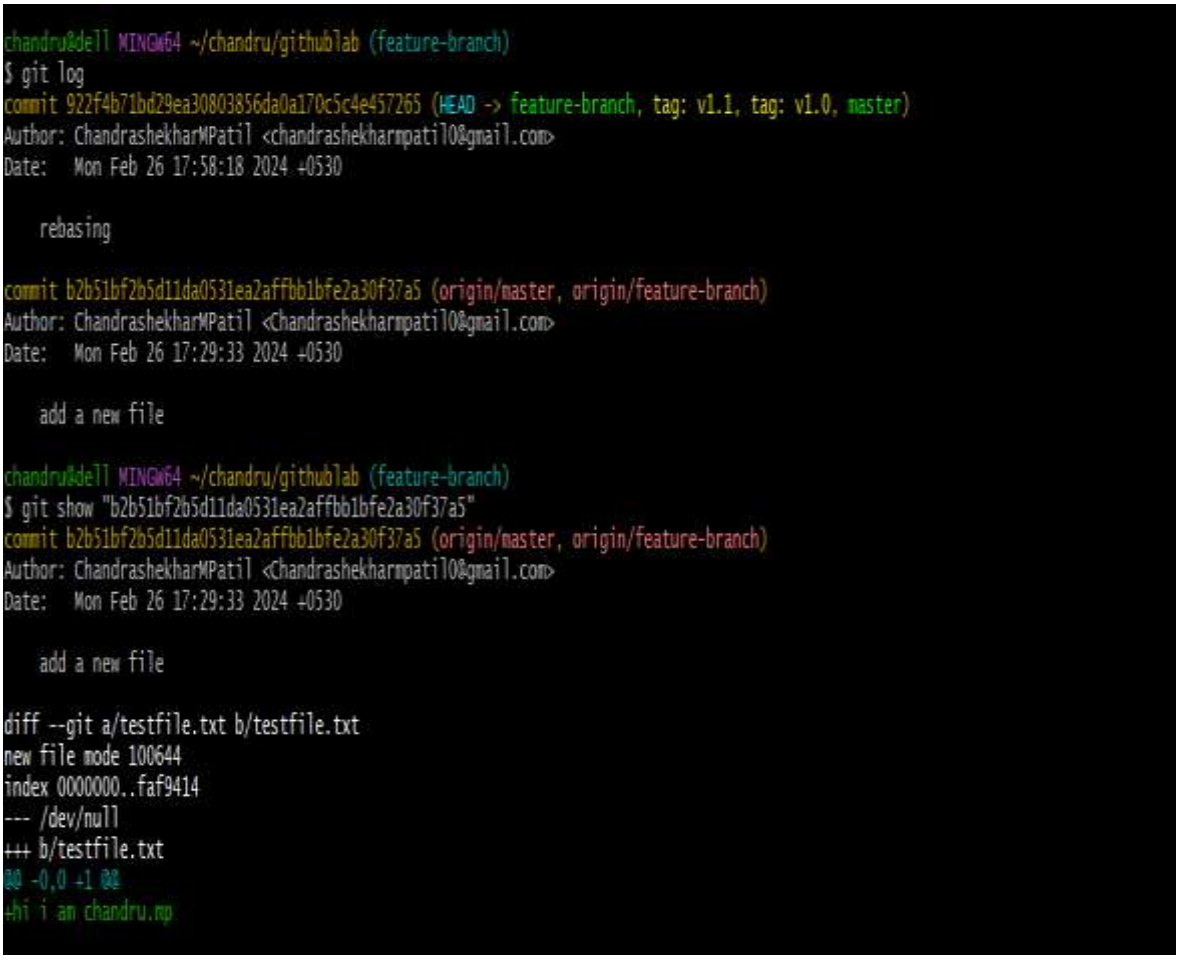
EXPERIMENT -09

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 **<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.



EXPERIMENT -10

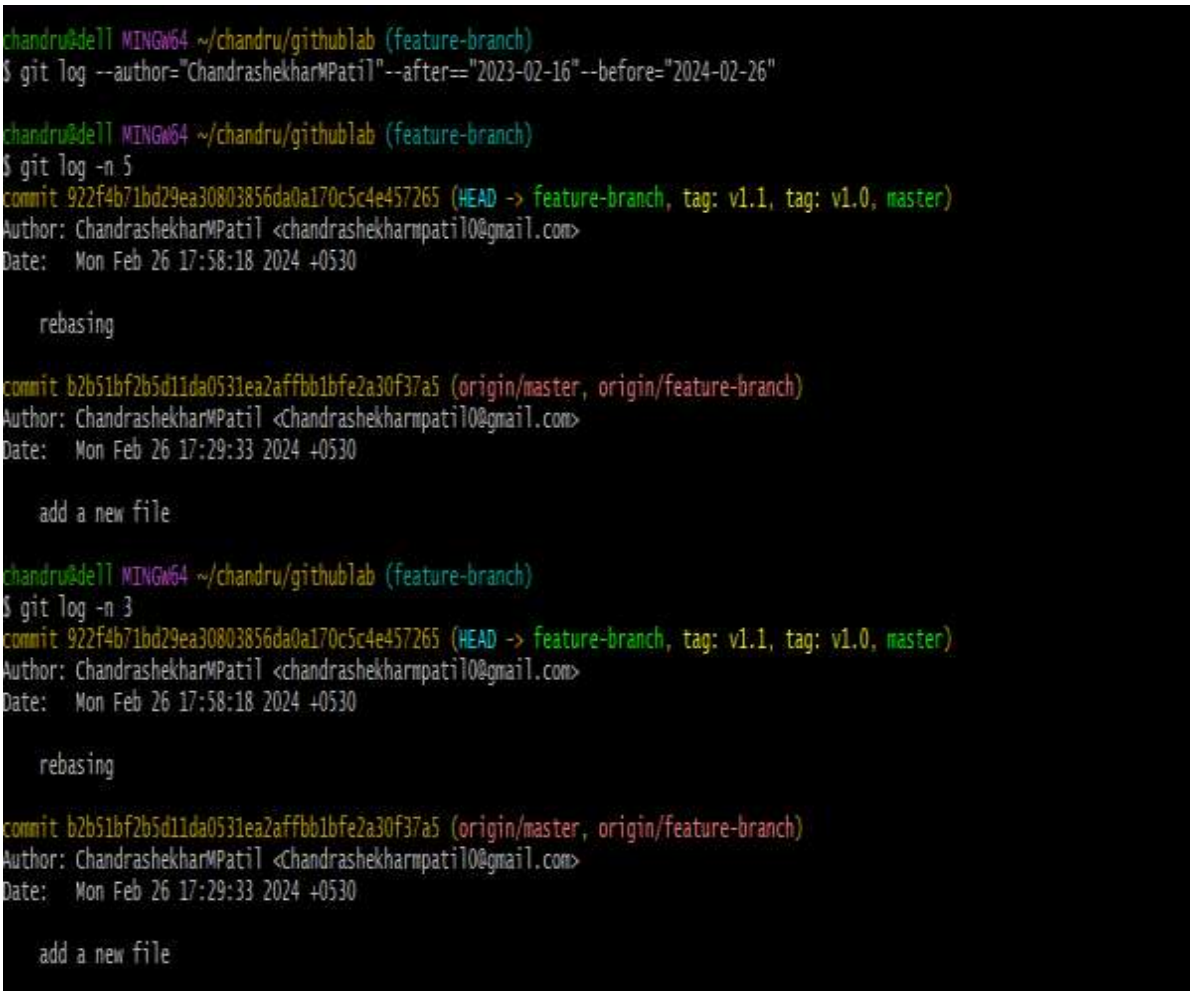
Analysing and changing Git History:

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 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.

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.

```
chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git log --author="ChandrashekharMPatil"--after=="2023-02-16"--before="2024-02-26"

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git log -n 5
commit 922f4b71bd29ea30803856da0a170c5c4e457265 (HEAD -> feature-branch, tag: v1.1, tag: v1.0, master)
Author: ChandrashekharMPatil <chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 17:58:18 2024 +0530

    rebasing

commit b2b51bf2b5d11da0531ea2affbb1bfe2a30f37a5 (origin/master, origin/feature-branch)
Author: ChandrashekharMPatil <Chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 17:29:33 2024 +0530

    add a new file

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git log -n 3
commit 922f4b71bd29ea30803856da0a170c5c4e457265 (HEAD -> feature-branch, tag: v1.1, tag: v1.0, master)
Author: ChandrashekharMPatil <chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 17:58:18 2024 +0530

    rebasing

commit b2b51bf2b5d11da0531ea2affbb1bfe2a30f37a5 (origin/master, origin/feature-branch)
Author: ChandrashekharMPatil <Chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 17:29:33 2024 +0530

    add a new file
```

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 **<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.

```
chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git revert "b2b51bf2b5d11da0531ea2affbb1bfe2a30f37a5"
[feature-branch 5cacb14] Revert "add a new file"
 1 file changed, 1 deletion(-)
 delete mode 100644 testfile.txt

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git revert "b2b51bf2b5d11da0531ea2affbb1bfe2a30f37a5" -m "work done"
error: option `mainline' expects a number greater than zero

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git log
commit 5cacb14ef32f5f8188acecf104317af6beeaf37a (HEAD -> feature-branch)
Author: ChandrashekharMPatil <chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 18:14:45 2024 +0530

    Revert "add a new file"

    This reverts commit b2b51bf2b5d11da0531ea2affbb1bfe2a30f37a5.

commit 922f4b71bd29ea30803856da0a170c5c4e457265 (tag: v1.1, tag: v1.0, master)
Author: ChandrashekharMPatil <chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 17:58:18 2024 +0530

    rebasing

commit b2b51bf2b5d11da0531ea2affbb1bfe2a30f37a5 (origin/master, origin/feature-branch)
Author: ChandrashekharMPatil <Chandrashekharmpatil0@gmail.com>
Date:   Mon Feb 26 17:29:33 2024 +0530

    add a new file

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$ git revert "b2b51bf2b5d11da0531ea2affbb1bfe2a30f37a5" -m "revert done"
error: option `mainline' expects a number greater than zero

chandru@dell MINGW64 ~/chandru/githublab (feature-branch)
$
```