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: AJAY GONHALMATH

USN: 1SV22CS003

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 **AJAY GONHALMATH** [**1SV22CS003**] 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 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.

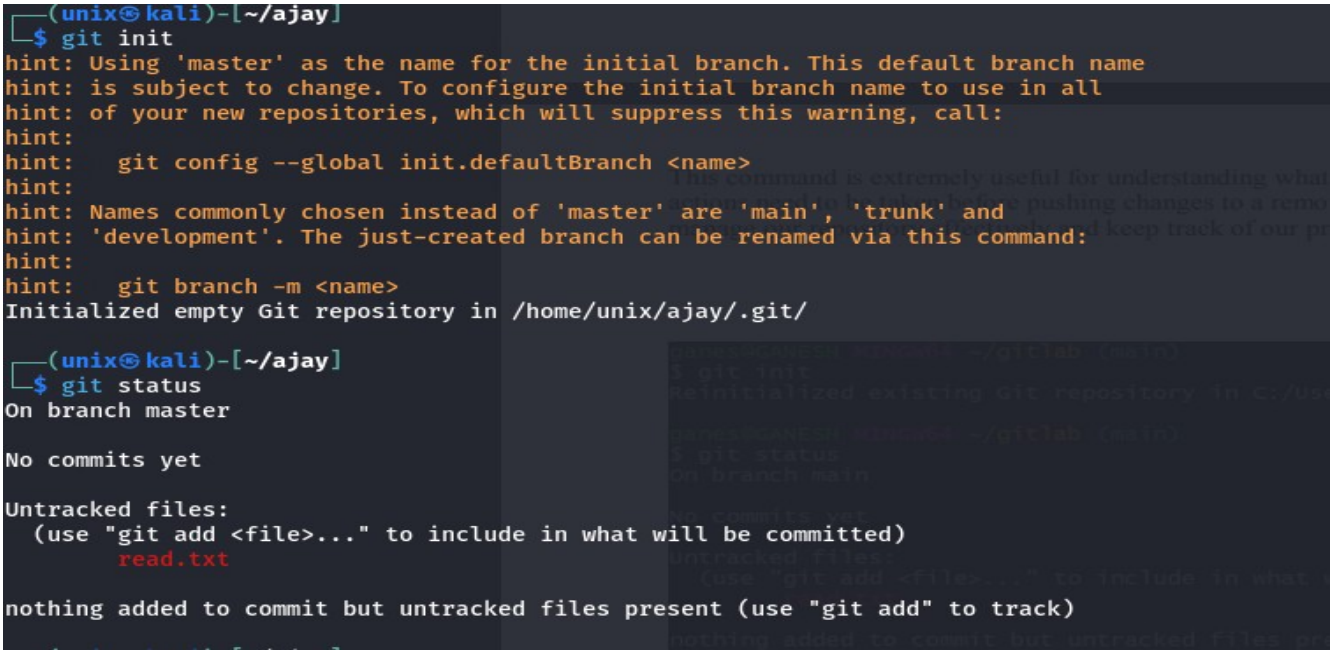
The above commands which are executed is shown below:

$ 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.



$ 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.

```
┌──(unix㉿kali)-[~/ajay]
└─$ git help
usage: 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>]

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

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 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.

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.

```
┌──(unix㉿kali)-[~/ajay]
└─$ git add read.txt
```

```
┌──(unix㉿kali)-[~/ajay]
└─$ git commit -m "commiting text file"
[master (root-commit) 6b0e2a4] commiting text file
 1 file changed, 1 insertion(+)
 create mode 100644 read.txt
```

$ 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 “ ajaygm18 ”

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 “ajayagm07@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 aremoterepositoryURLtoyour local Gitrepositorywith thename"origin."

This command establishes a connection between your local repository and a remoterepositoryhostedonaserver,suchasGitHuborGitLab.Theterm"origin"isaconventionally used name for the default remote repository, but you can choose any nameyouprefer.

- **$git pushoriginmaster:** Thecommand**gitpushoriginmaster**isusedtopushthecommitsfrom your local**master**branchtotheremoterepositorynamed**origin**.

Here'sabreakdownofwhateachpartofthecommand does:

- **git push**: This is the Git command used to push commits from your local repositorytoa remoterepository.
- **origin**: This refers to the name of the remote repository you're pushing to. In Gitterminology,"origin"isacommonnameusedtorefertothedefaultremoterepository.
- **master**: This refers to the local branch that you're pushing. In Git, "master" is thedefault nameforthe main branch ofarepository.

So,whenyourun**gitpushoriginmaster**,you'retellingGittopushthecommitsfromyourlocal**master**branchto theremoterepositorynamed**origin**.

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

```
┌──(unix㉿kali)-[~/ajay]
└─$ git config --global user.name "ajaygm18"

┌──(unix㉿kali)-[~/ajay]
└─$ git config --global user.email "ajayagm07@gmail.com"
```

```
┌──(unix㉿kali)-[~/ajay]
└─$ git remote add origin https://github.com/ajaygm18/gitlab.git
error: remote origin already exists.

┌──(unix㉿kali)-[~/ajay]
└─$ git push -u origin main
Username for 'https://github.com': ajaygm18
Password for 'https://ajaygm18@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 217 bytes | 108.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'main' on GitHub by visiting:
remote:      https://github.com/ajaygm18/gitlab/pull/new/main
remote:
To https://github.com/ajaygm18/gitlab
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

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.

$ **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.

$ **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

```
┌──(unix㉿kali)-[~/ajay]
└─$ git config --global user.name "ajaygm18"

┌──(unix㉿kali)-[~/ajay]
└─$ git config --global user.email "ajayagm17@gmail.com"
```

```
┌──(unix㉿kali)-[~/ajay]
└─$ git add .

┌──(unix㉿kali)-[~/ajay]
└─$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

$ **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.

$ **git checkout master** : used to switch to the master branch in your Git repository.

```
┌──(unix㉿kali)-[~/ajay]
└─$ git status
On branch feature-branch
nothing to commit, working tree clean

┌──(unix㉿kali)-[~/ajay]
└─$ 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/ajaygm18/gitlab/pull/new/feature-branch
remote:
To https://github.com/ajaygm18/gitlab
 * [new branch]      feature-branch -> feature-branch
```

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

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

$ **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.

```
┌──(unix㉿kali)-[~/ajay]
└─$ git log --oneline --decorate
6b0e2a4 (HEAD -> feature-branch, origin/master, origin/main, origin/feature-branch, master, main) commiting text file

┌──(unix㉿kali)-[~/ajay]
└─$ git merge feature-branch
Already up to date.

┌──(unix㉿kali)-[~/ajay]
└─$ git status
On branch feature-branch
nothing to commit, working tree clean
```

```
┌──(unix㉿kali)-[~/ajay]
└─$ git push origin master
Everything up-to-date
```

EXPERIMENT-03

Creating and Managing branches:

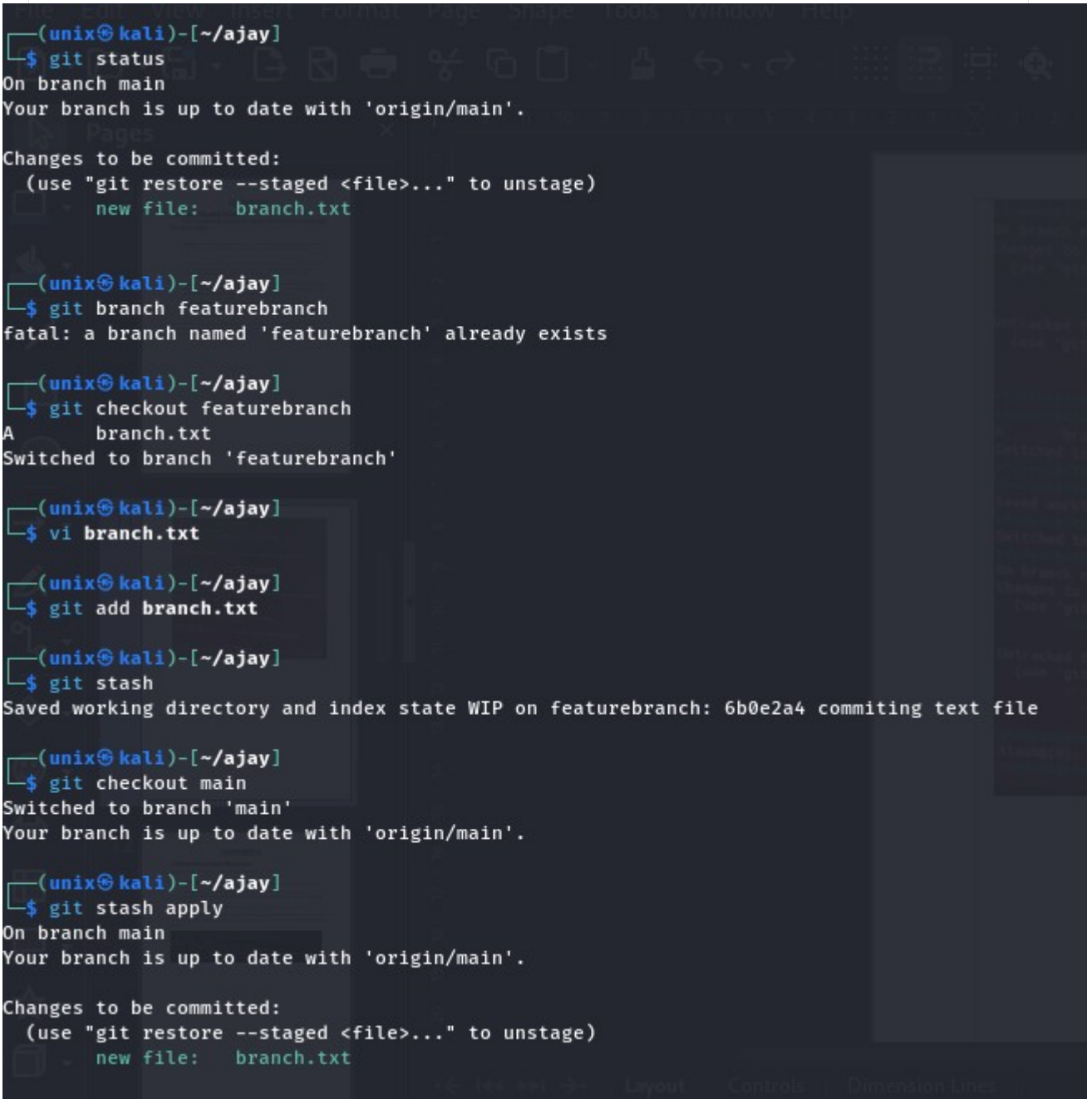
Write the commands to stash your changes ,switch branches and then apply the stashed changes.

1. **$gitstash:**commandisusedtotemporarilysavechangesinyourworkingdirectory and staging area so that you can work on something else or switch brancheswithoutcommittingthem.

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.

1. **$ git stash apply:**used to retrieve and reapply the most recent stash from the stashstack onto your current working directory. This command will reapply the changesfrom the stash ontoyour working directory without removing the stash from thestack.
2. **$ git stash list:**used to display the list of stashes in your Git repository's stash stack.Itshowsallthe stashesyou'vecreated,alongwithareference foreachstash

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.


┌──(unix㉿kali)-[~/ajay]
└─$ git status
On branch main
Your branch is up to date with 'origin/main'.
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
new file: branch.txt
┌──(unix㉿kali)-[~/ajay]
└─$ git branch featurebranch
fatal: a branch named 'featurebranch' already exists
┌──(unix㉿kali)-[~/ajay]
└─$ git checkout featurebranch
A branch.txt
Switched to branch 'featurebranch'
┌──(unix㉿kali)-[~/ajay]
└─$ vi branch.txt
┌──(unix㉿kali)-[~/ajay]
└─$ git add branch.txt
┌──(unix㉿kali)-[~/ajay]
└─$ git stash
Saved working directory and index state WIP on featurebranch: 6b0e2a4 commiting text file
┌──(unix㉿kali)-[~/ajay]
└─$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
┌──(unix㉿kali)-[~/ajay]
└─$ git stash apply
On branch main
Your branch is up to date with 'origin/main'.
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
new file: branch.txt

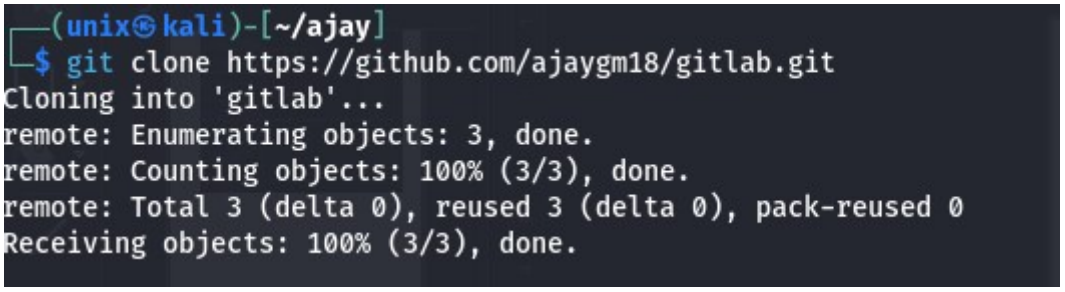
EXPERIMENT-04

Collaboration and remote Repositories:

Clone a remote git repository to your local machine.

$ git clone "repository URL" : The **gitclone** 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.
RepositoryURL 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 onyour local machine. You can then make changes, create commits, and push them back to theremoterepositoryas 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.

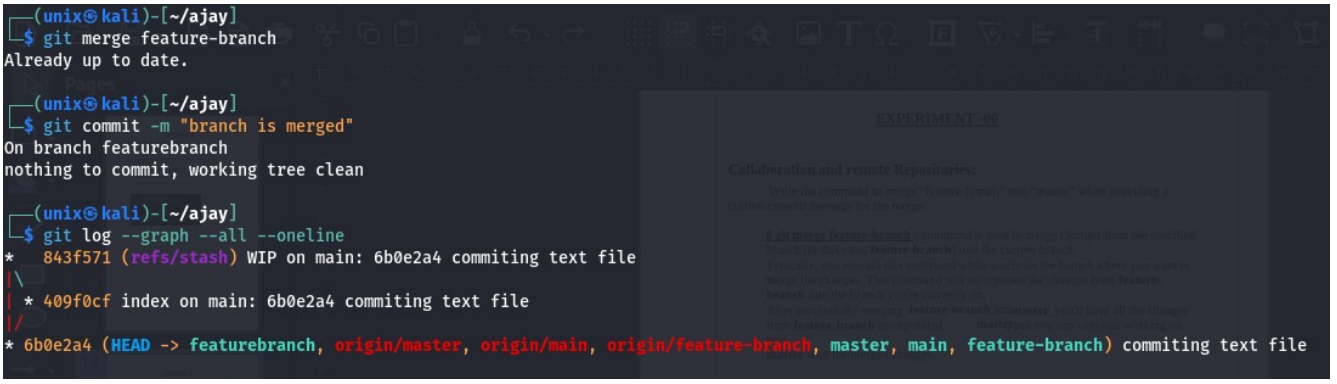
EXPERIMENT -06

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.



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.

1. **$ 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. Themessageprovidesadditional context orinformation 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.

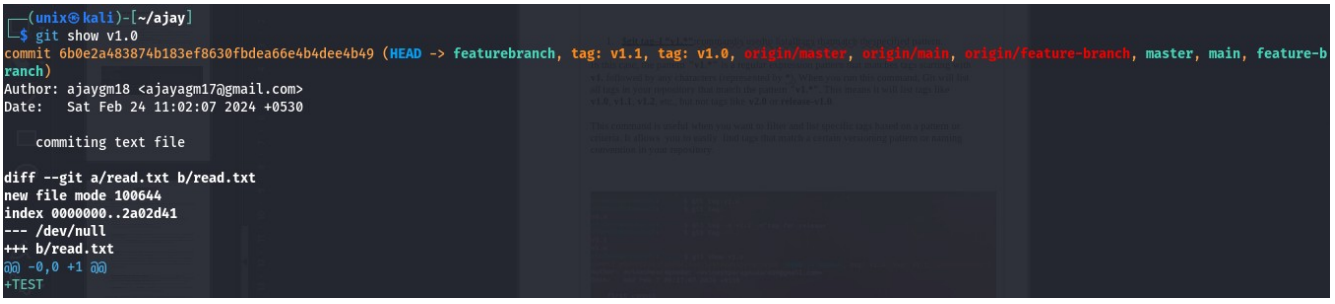
If **v1.0** is an annotated tag, the output will also include any additional metadata and the tag message. If it's a lightweight tag, the output will be similar to **git show** for a commit.

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.

1. **$git tag–l “v1.*”:**commandis usedto listalltags thatmatch thespecified 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.



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

```
┌──(unix㉿kali)-[~/ajay]
└─$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/unix/ajay/.git/

┌──(unix㉿kali)-[~/ajay]
└─$ vi 2.txt

┌──(unix㉿kali)-[~/ajay]
└─$ git add .

┌──(unix㉿kali)-[~/ajay]
└─$ git commit -m "1 commit"
[master (root-commit) b5d01c3] 1 commit
 1 file changed, 1 insertion(+)
 create mode 100644 2.txt

┌──(unix㉿kali)-[~/ajay]
└─$ vi 3.txt

┌──(unix㉿kali)-[~/ajay]
└─$ git add .

┌──(unix㉿kali)-[~/ajay]
└─$ git commit -m "2 commit"
[master 720664f] 2 commit
 1 file changed, 1 insertion(+)
 create mode 100644 3.txt

┌──(unix㉿kali)-[~/ajay]
└─$ git reflog
720664f (HEAD -> master) HEAD@{0}: commit: 2 commit
b5d01c3 HEAD@{1}: commit (initial): 1 commit

┌──(unix㉿kali)-[~/ajay]
└─$ git cherry-pick b5d01c3
On branch master
You are currently cherry-picking commit b5d01c3.
  (all conflicts fixed: run "git cherry-pick --continue")
  (use "git cherry-pick --skip" to skip this patch)
  (use "git cherry-pick --abort" to cancel the cherry-pick operation)

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

    git commit --allow-empty

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

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 command **git show** followed by the commit ID: **<commit_id>**

Replace about. with the actual commit ID you want to display information

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.

```
┌──(unix㉿kali)-[~/ajay]
└─$ git log
commit 720664f196b9d68dfc0d9f1f5156e5c70e33be48 (HEAD -> master)
Author: ajaygm18 <ajayagm17@gmail.com>
Date:   Sat Feb 24 11:40:33 2024 +0530

    2 commit

commit b5d01c3f15976f68b055dfa5ac9e7a067984d298
Author: ajaygm18 <ajayagm17@gmail.com>
Date:   Sat Feb 24 11:39:59 2024 +0530

    1 commit

┌──(unix㉿kali)-[~/ajay]
└─$ git show b5d01c3f15976f68b055dfa5ac9e7a067984d298
commit b5d01c3f15976f68b055dfa5ac9e7a067984d298
Author: ajaygm18 <ajayagm17@gmail.com>
Date:   Sat Feb 24 11:39:59 2024 +0530

    1 commit

diff --git a/2.txt b/2.txt
new file mode 100644
index 0000000..ca5441b
--- /dev/null
+++ b/2.txt
@@ -0,0 +1 @@
+dw
```

EXPERIMENT -10

Analysing and changing Git History:

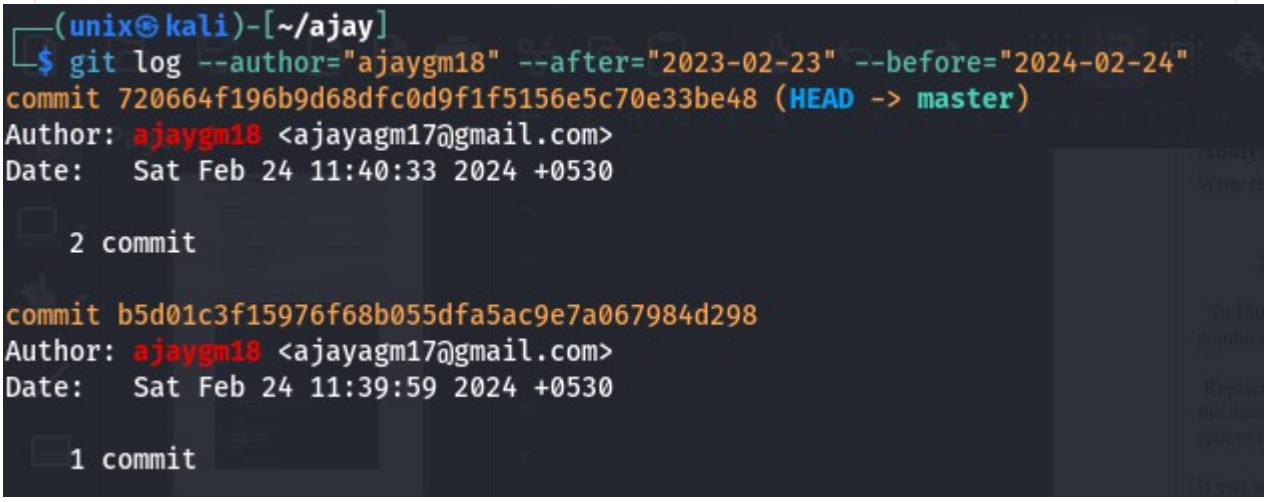
Write the command to list all commits made by author.

$ 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.

1. **$gitlog-n5:**Thiscommandisusedtodisplaythelast5commitsinyourrepository'scommit history.

It limits the output to the specified number of commits, in this case, 5. When you runthiscommand,Gitwilldisplaytheinformationforthelast5commitsinyourrepository,startingfromthe mostrecent commit andgoingbackwardin time.

This command is useful when you want to quickly view the most recent commits inyour repository,especiallyifyou'reonlyinterestedinaspecificnumber ofcommits.

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.

```
┌──(unix㉿kali)-[~/ajay]
└─$ git reflog
720664f (HEAD -> master) HEAD@{0}: commit: 2 commit
b5d01c3 HEAD@{1}: commit (initial): 1 commit

┌──(unix㉿kali)-[~/ajay]
└─$ git revert b5d01c3
[master bc44858] Revert "1 commit"
 Date: Sat Feb 24 11:39:59 2024 +0530
 1 file changed, 1 deletion(-)
 delete mode 100644 2.txt
```