Subject: Project Management with Git

Course Code: BCS358C

Name: Megharani Rajkumar Gani

USN: 4GW24CI026

Semester: 3rd Semester

Program 1

1. 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 appropriate commit message.

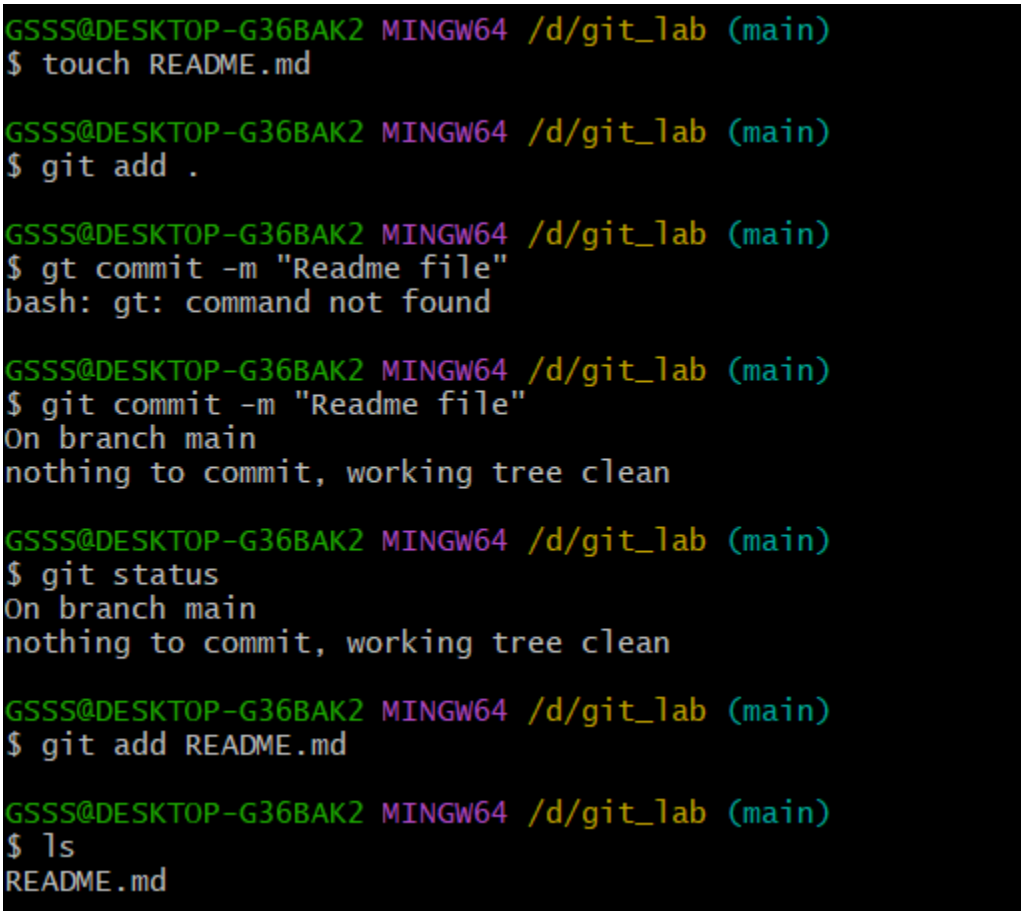
Program:

- git config --global user.name "gitmanual" – Sets the global Git username.
- mkdir project – Creates a new directory named project.
- cd project – Moves into the project directory.
- touch git.txt – Creates a new file named git.txt
- git add . – Adds all files to the staging area.
- git commit -m "Initial commit - adding a new file" – Saves the staged file to Git with a commit message.

```
GSSS@DESKTOP-G36BAK2 MINGW64 ~
$ git config --global user.name "gitmanual"

GSSS@DESKTOP-G36BAK2 MINGW64 ~
$ mkdir project

GSSS@DESKTOP-G36BAK2 MINGW64 ~
$ cd project

GSSS@DESKTOP-G36BAK2 MINGW64 ~/project
$ git init
Initialized empty Git repository in C:/Users/GSSS/project/.git/

GSSS@DESKTOP-G36BAK2 MINGW64 ~/project (main)
$ touch git.txt

GSSS@DESKTOP-G36BAK2 MINGW64 ~/project (main)
$ git add .

GSSS@DESKTOP-G36BAK2 MINGW64 ~/project (main)
$ git commit -m "Initial commit - adding a new file"
[main (root-commit) 7d01e21] Initial commit - adding a new file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 git.txt
```

Program 2: Creating and Managing Branches Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."

We have to initialize our git directory and add a file in the main directory.



Now we have to create a branch named feature-branch. We use ‘branch’ command.

```
GSSS@DESKTOP-G36BAK2 MINGW64 /d/git_lab (feature-branch)
$ git branch feature-branch
```

The command is: git branch feature-branch

Again the same procedure we add a file there. Now we have to get out of the branch and we have to switch to the main branch. For that we have to use the ‘checkout’ command.

The command is: git checkout main

To go back to the feature-branch we have to use: git checkout feature-branch

```
GSSS@DESKTOP-G36BAK2 MINGW64 /d/git_lab (master)
$ git checkout main
Switched to branch 'main'
```

Now we use the ‘merge’ command to merge both the branches.

git merge feature-branch

```
GSSS@DESKTOP-G36BAK2 MINGW64 /d/git_lab (main)
$ git merge feature-branch
Updating c964361..a5786f8
Fast-forward
 README.md | 1 -
 file1.txt | 0
 2 files changed, 1 deletion(-)
 create mode 100644 file1.txt

GSSS@DESKTOP-G36BAK2 MINGW64 /d/git_lab (main)
$
```

We can see the all the files in both the branches have been merged to the main branch.

Program 3: Creating and Managing Branches Write the commands to stash your changes, switch branches, and then apply the stashed changes.

So in this program we are supposed to create a branch, make some changes, switch to the new branch and the changes which we have made in the old branch, we are supposed to stash it in the new branch. For this:-

Create a new file and add the file to your local repository. The commands for these functions have been given in the previous programs. Then before committing we are supposed to temporarily save our work, for this we have to give the command,

```
git stash push -m "WIP: my changes"
```

```
git checkout -b target-branch
```

```
git stash pop
```

```
admin@DESKTOP-ENRDLON MINGW64 /d/usn/New fo
lder (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2
commits.
  (use "git push" to publish your local com
mits)

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

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

admin@DESKTOP-ENRDLON MINGW64 /d/usn/New fo
lder (main)
$ git add .

admin@DESKTOP-ENRDLON MINGW64 /d/usn/New fo
lder (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2
commits.
  (use "git push" to publish your local com
mits)

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


admin@DESKTOP-ENRDLON MINGW64 /d/usn/New fo
lder (main)
$ git stash push -m"WIP: my cahnges"
Saved working directory and index state On
main: WIP: my cahnges

admin@DESKTOP-ENRDLON MINGW64 /d/usn/New fo
lder (main)
$ git checkout -b target-branch
Switched to a new branch 'target-branch'

admin@DESKTOP-ENRDLON MINGW64 /d/usn/New fo
lder (target-branch)
$ git stash pop
On branch target-branch
Changes to be committed:
```

```
admin@DESKTOP-ENRDLON MINGW64 /d/usn/New fo
lder (target-branch)
$ git stash pop
On branch target-branch
Changes to be committed:
  (use "git restore --staged <file>..." to
unstage)
        new file:   file1.txt

Dropped refs/stash@{0} (0100c5e91ca18679496
c71a9f1e6bc4a83755fe3)

admin@DESKTOP-ENRDLON MINGW64 /d/usn/New fo
lder (target-branch)
$ git status
On branch target-branch
Changes to be committed:
  (use "git restore --staged <file>..." to
unstage)
        new file:   file1.txt
```

Program 4: Collaboration and Remote Repositories Clone a remote Git repository to your local machine.

In github I had created a repository under my USN. So we are going to clone this repository in my system

git clone https://github.com/meghagani21@gmail.com”

```
admin@DESKTOP-ENRDLON MINGW64 /d/usn/New folder (master)
$ git clone https://github.com/meghagani21-png/megha1.git
Cloning into 'megha1'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
```

Program 5: Collaboration and Remote Repositories Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

So with the repository we have created we have to fetch the changes we have made in origin/main, create and switch to a branch(target-branch) and the changes made in the main branch let us bring it to the target-branch. We have to create a new file in the target-branch.

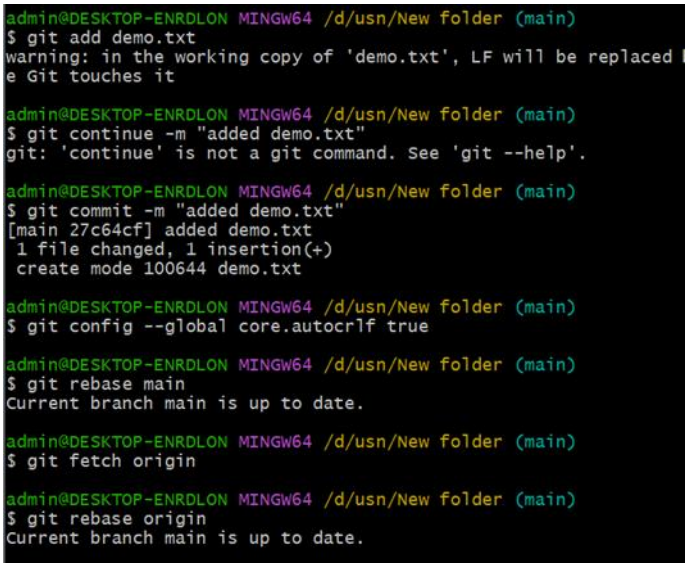
If there are updates in the main branch of the remote repository we can get it using the 'fetch' command.

git fetch origin

Then we switch to the target-branch. Then we can rebase our local branch onto remote.

git checkout

git rebase origin/main

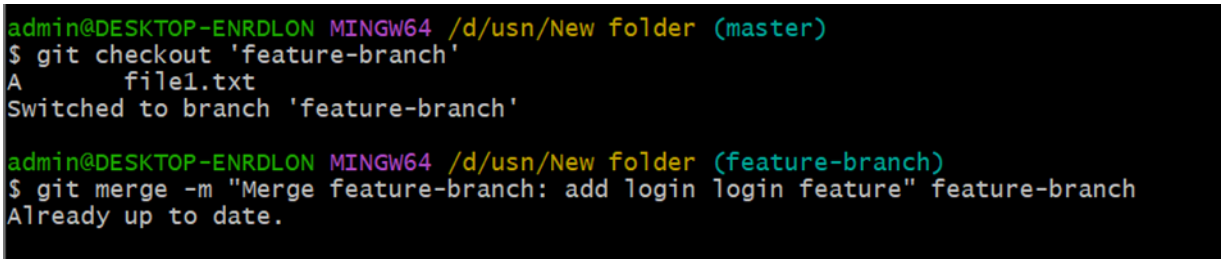


Program 6: Collaboration and Remote Repositories Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

To merge "feature-branch" into "master" with a custom commit message, first switch to master, then use git merge -m "your message" feature-branch. This creates a merge commit preserving both branch histories. This command ensures you're on the target branch.

git checkout master

git merge -m "Merge feature-branch: add login feature" feature-branch

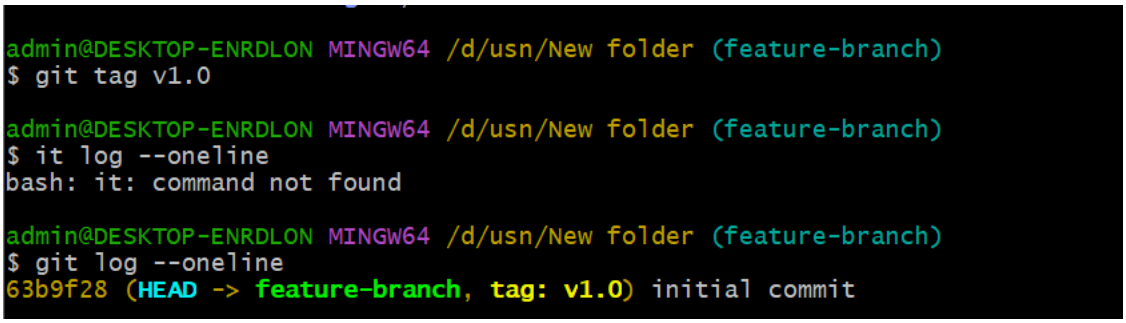


Program 7: Git Tags and Releases Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.

A lightweight Git tag is a simple pointer (reference) to a specific commit, like a bookmark with no extra metadata. This behaves like a non-moving branch. The command creates a lightweight tag pointing to your current commit.

git tag v1.0

git log –oneline



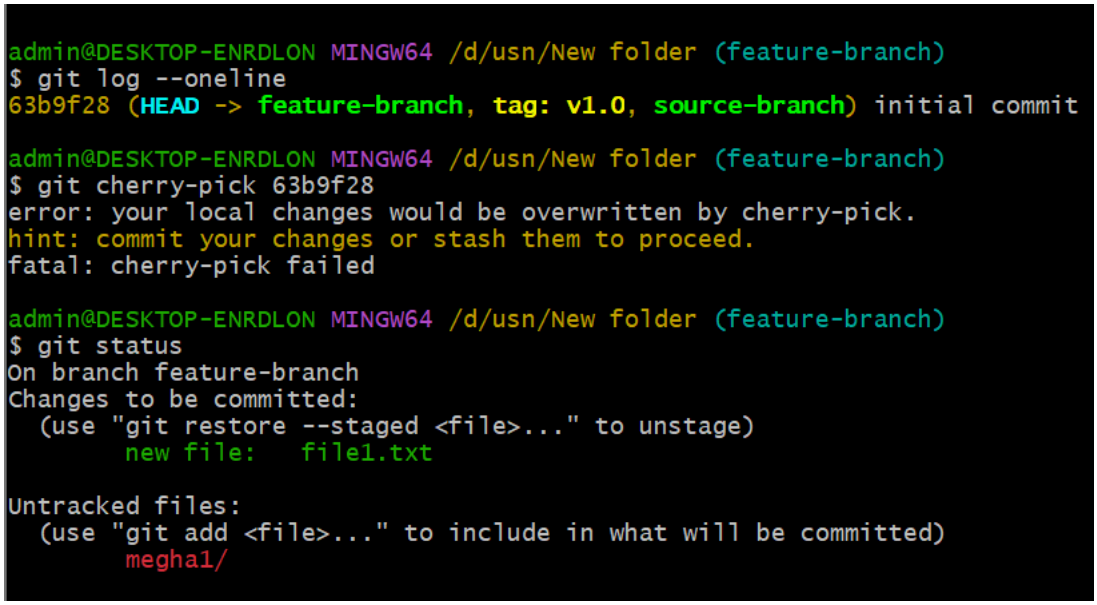
Program 8: Advanced Git Operations: Write the command to cherry-pick a range of commits from "source-branch" to the current branch

So here I have created a branch named ‘source-branch’. I have added two files and made modifications in one file. I have added the files and committed the changes. Then we go to the master branch. Then we check the log

git log source-branch --oneline -10

Using this command, we can see the commits with their id’s. After this we use he ‘cherry-pick’ command. We give the range from the oldest id to the newest id.

git cherry-pick



```
admin@DESKTOP-ENRDLON MINGW64 /d/usn/New folder (feature-branch)
$ git add "megha1/"

admin@DESKTOP-ENRDLON MINGW64 /d/usn/New folder (feature-branch)
$ git commit -m"my local changes"
[feature-branch a40cb55] my local changes
 2 files changed, 1 insertion(+)
 create mode 100644 file1.txt
 create mode 160000 megha1

admin@DESKTOP-ENRDLON MINGW64 /d/usn/New folder (feature-branch)
$ git cherry-pick 63b9f28
On branch feature-branch
You are currently cherry-picking commit 63b9f28.
  (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'

admin@DESKTOP-ENRDLON MINGW64 /d/usn/New folder (feature-branch|CHERRY-PICKING)
$ git cherry-pick --abort
```

Program 9: 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?

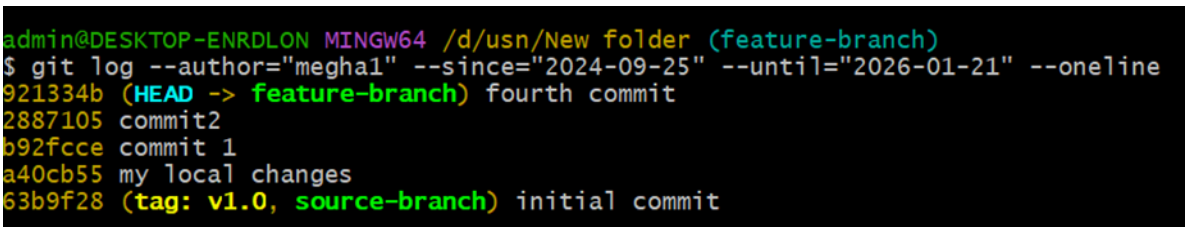
To view the history and all other details we use only one command that is the ‘show’ command.

git show abc1234

Program 10: Analysing and Changing Git History Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."

git log --author="megha1" --since="2023-01-01" --until="2023-12-31"

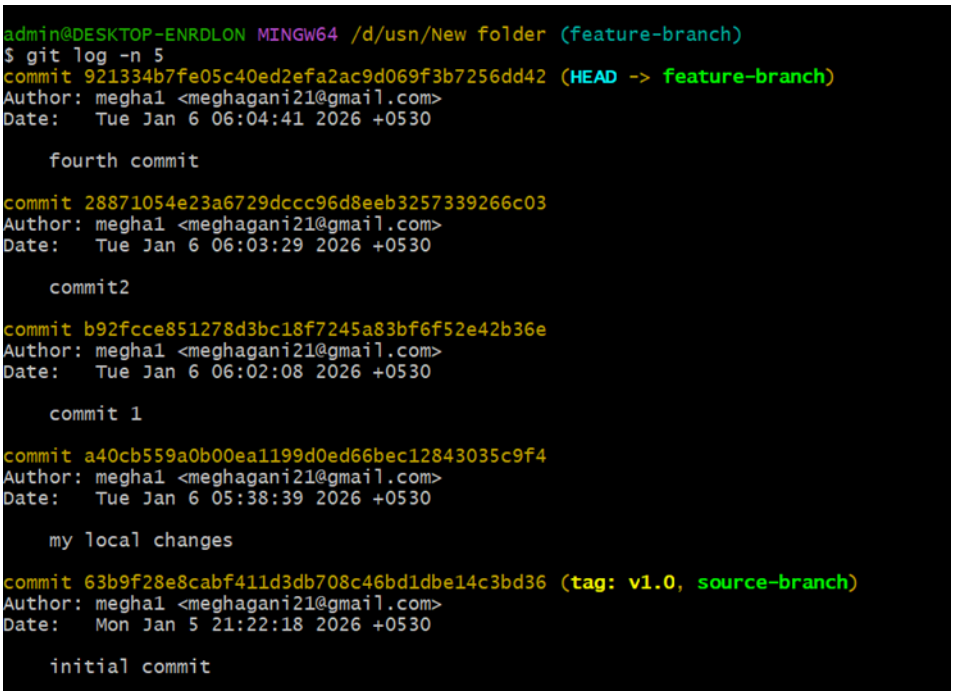
This command will display a list of commits made by the author "megha1" that fall within the specified date range, from January 1, 2023, to December 31, 2023. Make sure to adjust the author's



Program 11: Analysing and Changing Git History Write the command to display the last five commits in the repository's history.

To display the last five commits in a Git repository's history, you can use the git log command with the -n option, which limits the number of displayed commits. Here's the command:

```
git log -n 5
```



Program 12: Analyzing and Changing Git History Write the command to undo the changes introduced by the commit with the ID "abc123".

To undo the changes introduced by a specific commit with the ID "abc123" in Git, you can use the git revert command. The git revert command creates a new commit that undoes the changes made by the specified commit, effectively "reverting" the commit. Here's the command:

```
git revert abc123
```

Replace "abc123" with the actual commit ID that you want to revert. After running this command, Git will create a new commit that negates the changes introduced by the specified commit. This is a safe way to undo changes in Git because it preserves the commit history and creates a new commit to record the reversal of the changes.

```
[feature-branch f2ea364] Revert "fourth commit"
 1 file changed, 1 insertion(+), 1 deletion(-)
```