Experiment 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 an appropriate commit message.

→ To create a new directory
$ mkdir "filename"

→ To open / move into that directory
$ cd "filename"

→ To configure username
$ git config --global user.name " , "
give the username used for - git login

→ To configure email id
$ git config --global user.email " email id "

→ To configure password
$ git config --global user.password " password "

→ To check the configuration details
$ git config --global --list

→ Open your terminal and navigate to the directory where you want to create the git repository

→ Initialize a new git repository in that directory
~~$ git add~~ $ git init

→ Create a new file in the directory and add it to the staging area

$ git add .

→ Commit the changes with an appropriate commit message

$ git commit -m "Your message here"

→ Your changes will be committed ~~to~~ to the Git repository with the provided commit message.

2 Creating & Managing Branches

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

1. Make sure you are in the "master" branch by switching to it

$ git ~~check~~ checkout master

2. Create a new branch named "feature-branch" and switch to it:

[~~$ git checkout -b feature-branch~~] ×

$ git branch <branch name>

$ git branch feature-branch

Above command will create a branch

$ git checkout -b feature-branch

Above command will create and check out a new branch at the same time

~~$ get~~ $ git branch -d <branch name>

Deletes a specified branch

$ git branch -D <branch name>

Forces the deletion of a specified branch even if it has unmerged changes

$ git branch -m <branch name>

Rename the current branch

$ git branch -a

Lists all remote branches

(3) After creating and switching to the new branch, make your changes by adding, deleting the files as needed.

(4) Stage and commit your changes

(5) ~~Switch back to the "master" branch~~

$ git add.

$ git commit -m "Your message"

(5) Switch back to the "master" branch

$ git checkout master

(6) Merge the "feature-branch" into the "master" branch

$ git merge feature-branch

(7) To delete the branch from remote repository

git push origin -delete "branchname"

3 Creating and Managing Branches

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

-> Git stash is used when the work is incomplete in one branch but you want to ~~start~~ ~~in~~ do some changes in other branches then you can use.

-> If you switch branch without committing Two things will happen

1. Switches to the branch carrying the changes
2. Git will not allow to switch the branch and asks to commit or stash the changes

The stash's meaning is "store something safely in a hidden place". saves your data safely without committing

The git stash command enables you to switch branches without committing the current branch

Step1: Create new project & git stash save "message"

$ git add.

$ git commit

$ git stash : To do stash

git stash list : stash id

To retrieve the data from stash

$ git stash pop

Exp: 6: Collaboration & Remote Repositories

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

Step 1:

Go to git bash

↓

Create branch by the name "feature-branch"

$ git branch feature-branch

Step 2: Checkout to master branch

$ git checkout master

Step 3: Merge feature branch with master branch using commit message

$ git merge feature-branch -m "Your custom message"

Exp : 4

Collaboration and Remote Repositories

Clone a remote Git repository to your local machine

1. Open your terminal / command prompt
2. Navigate to the directory where you want to clone the remote git repository
3. Use the below command to clone the remote repository

$ git clone <repository_url>

Ex: $ git clone https://github.com/usrname/repo_name.git

4. Git will clone the repository to your local machine
5. Now you can work with the cloned repository on your local machine, make change and push the changes to the remote repository

$ git push -u origin master

or

$ git push -u "copy the url" master

Expt - 7

Git Tag

Write the command to create a lightweight git tag named "V1.0" for a commit in your local repository

Step 1:

On your system create one folder → go inside the folder → open git bash → Next checkout the branch where u want to create the tag

$ git branch : List all the available branches

$ git checkout <branch name>

Ex: git checkout master

Step 2: Create one or two files and commit it

$ touch filename

$ ~~touch~~ ~~adds~~ git add .

$ git commit -m "type some message"

Step 3: Create tag

$ git tag <tag name>

Ex: $ git tag V1.0 : light weight tag

Ex: $ git tag -a V1.1 -m "tag for release"

annoted tag, contains some message and stored as a object in a repository

$ git tag : give the list of tag created

$ git show V1.0

Step 4: Push tags to remote repository

git push origin V1.0 : Push only one tag
git push --tags : Push all the tag

Step 5: To Delete the tags on local repository

$ git tag -d V1.0

Step 6: To Delete tags from remote repository

$ git push origin -d V1.0

To delete multiple tags
$ git tag -d V1.0 V1.1 (local)
$ git push origin -d V1.0 V1.1 (remote)

To create a tag using commit ID
$ git tag <tag name> <commit id>
Ex: $ git tag V1.2 5f8346

Exp-8: Advanced Git Operations

(6)

GIT Cherry Pick

→ A command that allows you to select and apply a single commit from one branch to another

When to use cherry-picking

→ Apply the same changes to multiple branches

→ Apply specific changes before creating or merging a pull request

→

How to use cherry-picking

1. Navigate to the folder that contains your git repo

2. Check out the branch you want to work on

$ git branch 1.0
$ git branch 2.0
$ git branch 3.0
} create 3 branches

git log --oneline (To find the hash for a commit in the commit history

U have committed one file in branch 3.0, now u want to apply the commit to other branches

3. Find the hash for the commit you want to cherry-pick and checkout to the branch u want on

Currently we are in branch 3.0, now want to do apply the changes in branch 2.0. First checkout to branch 2.0

$ git checkout 2.0

Next use cherry-pick command

$ git cherry-pick <commit hash>

Next checkout to branch 1.0

$ git checkout 1.0

Next use cherry-pick command

$ git cherry-pick <commit hash>

Exp-9

Analysing & Creating Git History

Given a commit ID. how would you use git to view the details of the specific commit, including the author, date & commit message.

Sol: To view the details of specific commit, which includes the author, date & commit message.

In git bash:

$ git show <commit-ID>

Ex: $ git show abc123

Exp: 10 Analysing & changing Git History

Write the command to list all commits made by the author "John Doe" b/w "2023-01-01" and "2023-12-31"

Soln: To list all commits made by the specific author from start date to end date

```
$ git log --author="JohnDoe"--since="2023-01-01")
          --untill ="2023-12-31")
```

Exp11: Analysing and changing Git History

Write the command to display the last 5 commits in the repository's history

```
$ git log -n1 <commit ID>
```

-n1 = tells the git to show only one commit

Exp12: Write the command to undo the changes introduced by the commit with ID "abc123"

: To undo the changes from the specific commit u can use revert command

```
$ git revert abc123
```

Once u give this command it will ask you to type the message in new window

pause => control + C

Type : q a → Enter

Exp: 5: Collaboration & Remote Repositories

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

Step 1: Create a repository in git hub

Step 2: Create one folder on your system and clone from the repository you have created

$ git clone "paste url"

Enter into the folder which you have cloned and open git bash

Step 3: Create new branch

$ git feature $ git checkout feature

Step 4: Create 2 files in feature branch

$ touch f1.txt

$ git add .

$ git commit -m "f1 added"

$ touch f2.txt

$ git add.

$ git commit -m "f2 added"

Step 5: Switch to main branch

$ git branch : will show the list of branches

$ git checkout main

Step 6: Create a file in main/master branch

$ touch m1.txt

$ git add .

$ git commit -m "m1 added"

Step 7: Now rebase the files from feature branch

$ git rebase feature

$ git log : will show the list

(8)

git rebase is a command that integrates changes from one branch into another, rewriting the project history

git merge: combines changes from one branch into another branch.

git rebases moves the changes from one branch to another branch.