Jawaharlal Nehru New College of Engineering

NAVULE (SAVALANGA Road), Shivamogga – 577204. Karnataka

Department of Artificial Intelligence and Machine Learning

PROJECT MANAGEMENT WITH GIT

(BCS358C) ,

III Sem

LAB MANUAL

(2023-24)

Prepared by

Mr. RANJAN V

Assistant Professor,

Dept. of AIML,

JNNCE Shivamogga

Course Objectives:

- To familiar with basic command of Git
- To create and manage branches
- To understand how to collaborate and work with Remote Repositories
- To familiar with vision controlling commands

Course Outcomes:

At the end of the course the student will be able to:
● Use the basics commands related to git repository
● Create and manage the branches
● Apply commands related to Collaboration and Remote Repositories
● Use the commands related to Git Tags, Releases and advanced git operations
● Analyse and change the git history

Textbooks:

● Version Control with Git, 3rd Edition, by Prem Kumar Ponuthorai, Jon Loeliger Released October 2022, Publisher(s): O'Reilly Media, Inc.
● Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, https://gitscm. com/book/en/v2
●https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_013094443347369984278 2_shared/overview
●https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_013301347121774592119 26_shared/overview

GIT BASICS

What is Git?

Git is a distributed version control system (VCS) that is widely used for tracking changes in source code during software development. It was created by Linus Torvalds in 2005 and has since become the de facto standard for version control in the software development industry.
Git allows multiple developers to collaborate on a project by providing a history of changes, facilitating the tracking of who made what changes and when. Here are some key concepts and features of Git:

1. Repository (Repo): A Git repository is a directory or storage location where your project's files and version history are stored. There can be a local repository on your computer and remote repositories on servers.
2. Commits: In Git, a commit is a snapshot of your project at a particular point in time. Each commit includes a unique identifier, a message describing the changes, and a reference to the previous commit.
3. Branches: Branches in Git allow you to work on different features or parts of your project simultaneously without affecting the main development line (usually called the "master" branch). Branches make it easy to experiment, develop new features, and merge changes back into the main branch when they are ready.
4. Pull Requests (PRs): In Git-based collaboration workflows, such as GitHub or GitLab, pull requests are a way for developers to propose changes and have them reviewed by their peers. This is a common practice for open-source and team-based projects.
5. Merging: Merging involves combining changes from one branch (or multiple branches) into another. When a branch's changes are ready to be incorporated into the main branch, you can merge them.
6. Remote Repositories: Remote repositories are copies of your project stored on a different server. Developers can collaborate by pushing their changes to a remote repository and pulling changes from it. Common remote repository hosting services include GitHub, GitLab, and Bitbucket.
7. Cloning: Cloning is the process of creating a copy of a remote repository on your local machine. This allows you to work on the project and make changes locally.
8. Forking: Forking is a way to create your copy of a repository, typically on a hosting platform like GitHub. You can make changes to your fork without affecting the original project and later create pull requests to contribute your changes back to the original repository.

What is Version Control System (VCS)?

A Version Control System (VCS), also commonly referred to as a Source Code Management (SCM) system, is a software tool or system that helps manage and track changes to files and directories over time. The primary purpose of a VCS is to keep a historical record of all changes made to a set of files, allowing multiple people to collaborate on a project while maintaining the integrity of the codebase.

There are two main types of VCS: centralized and distributed.

Centralized Version Control Systems (CVCS): In a CVCS, there is a single central repository that stores all the project files and their version history. Developers check out files from this central repository, make changes, and then commit those changes back to the central repository. Examples of CVCS include CVS (Concurrent Versions System) and Subversion (SVN).

Distributed Version Control Systems (DVCS): In a DVCS, every developer has a complete copy of the project's repository, including its full history, on their local machine. This allows developers to work independently, create branches for experimentation, and synchronize their changes with remote repositories. Git is the most well-known and widely used DVCS, but other DVCS options include Mercurial and Bazaar.

Git Installation

To install Git on your computer, you can follow the steps for your specific operating system:

1. Installing Git on Windows:

a. **Using Git for Windows (Git Bash):**

- Go to the official Git for Windows website: https://gitforwindows.org/
- Download the latest version of Git for Windows.
- Run the installer and follow the installation steps. You can choose the default settings for most options.

b. Using GitHub Desktop (Optional):

- If you prefer a graphical user interface (GUI) for Git, you can also install GitHub Desktop, which includes Git. Download it from https://desktop.github.com/ and follow the installation instructions.

To download

https://git-scm.com/download/win

2. Installing Git from Source (Advanced):

- If you prefer to compile Git from source, you can download the source code from the official Git website (https://git-scm.com/downloads) and follow the compilation instructions provided there. This is usually only necessary for advanced users.

After installation, you can open a terminal or command prompt and verify that Git is correctly installed by running the following command:

$ git --version

If Git is installed successfully, you will see the Git version displayed in the terminal. You can now start using Git for version control and collaborate on software development projects.

Git Commands List

Git is a popular version control system used for tracking changes in software development projects. Here's a list of common Git commands along with brief explanations:

1. **git init**: Initializes a new Git repository in the current directory.
2. **git clone <repository URL>**: Creates a copy of a remote repository on your local machine.
3. **git add <file>**: Stages a file to be committed, marking it for tracking in the next commit.
4. **git commit -m "message"**: Records the changes you've staged with a descriptive commit message.

5. **git status**: Shows the status of your working directory and the files that have been modified or staged.
6. **git log**: Displays a log of all previous commits, including commit hashes, authors, dates, and commit messages.
7. **git diff**: Shows the differences between the working directory and the last committed version.
8. **git branch**: Lists all branches in the repository and highlights the currently checked-out branch.
9. **git branch <branchname>**: Creates a new branch with the specified name.
10. **git checkout <branchname>**: Switches to a different branch.
11. **git merge <branchname>**: Merges changes from the specified branch into the currently checked-out branch.
12. **git pull**: Fetches changes from a remote repository and merges them into the current branch.
13. **git push**: Pushes your local commits to a remote repository.
14. **git remote**: Lists the remote repositories that your local repository is connected to.
15. **git fetch**: Retrieves changes from a remote repository without merging them.
16. **git reset <file>**: Unstages a file that was previously staged for commit.
17. **git reset --hard <commit>**: Resets the branch to a specific commit, discarding all changes after that commit.
18. **git stash**: Temporarily saves your changes to a "stash" so you can switch branches without committing or losing your work.
19. **git tag**: Lists and manages tags (usually used for marking specific points in history, like releases).
20. **git blame <file>**: Shows who made each change to a file and when.
21. **git rm <file>**: Removes a file from both your working directory and the Git repository.
22. **git mv <oldfile> <newfile>**: Renames a file and stages the change.

These are some of the most common Git commands, but Git offers a wide range of features and options for more advanced usage. You can use git --help followed by the command name to get more information about any specific command, e.g., git help commit.

Experiments On

Project Management with Git
(As Per VTU Syllabus)

Experiment 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.

Solution:
To initialize a new Git repository in a directory, create a new file, add it to the staging area, and commit the changes with an appropriate commit message, follow these steps:

1. Open your terminal and navigate to the directory where you want to create the Git repository.
2. Initialize a new Git repository in that directory:

$ git init

1. Create a new file in the directory. For example, let's create a file named "my_file.txt." You can use any text editor or command-line tools to create the file.
2. Add the newly created file to the staging area. Replace "my_file.txt" with the actual name of your file:

$ git add my_file.txt

This command stages the file for the upcoming commit.

1. Commit the changes with an appropriate commit message. Replace "Your commit message here" with a meaningful description of your changes:

$ git commit -m "Your commit message here"
Your commit message should briefly describe the purpose or nature of the changes you made. For

example:

$ git commit -m "Add a new file called my_file.txt"

After these steps, your changes will be committed to the Git repository with the provided commit message. You now have a version of the repository with the new file and its history stored in Git.

Experiment 2.

Creating and Managing Branches:

Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."
Solution:
To create a new branch named "feature-branch," switch to the "master" branch, and merge the "feature-branch" into "master" in Git, follow these steps:

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

$ git checkout master

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

$ git checkout -b feature-branch

This command will create a new branch called "feature-branch" and switch to it.
1. Make your changes in the "feature-branch" by adding, modifying, or deleting files as needed.
2. Stage and commit your changes in the "feature-branch":

$ git add .
$ git commit -m "Your commit message for feature-branch"

Replace "Your commit message for feature-branch" with a descriptive commit message for the changes you made in the "feature-branch."
1. Switch back to the "master" branch:

$ git checkout master

1. Merge the "feature-branch" into the "master" branch:

$ git merge feature-branch

This command will incorporate the changes from the "feature-branch" into the "master" branch.
Now, your changes from the "feature-branch" have been merged into the "master" branch. Your project's history will reflect the changes made in both branches

```
Ranjith@Ranjith-PC MINGW32 ~ (master)
$ cd desktop

Ranjith@Ranjith-PC MINGW32 ~/desktop (master)
$ cd gitlab

Ranjith@Ranjith-PC MINGW32 ~/desktop/gitlab (master)
$ git checkout master
Already on 'master'

Ranjith@Ranjith-PC MINGW32 ~/desktop/gitlab (master)
$ git checkout master
Already on 'master'

Ranjith@Ranjith-PC MINGW32 ~/desktop/gitlab (master)
$ ^[[200~git checkout -b feature-branch
bash: $'\E[200~git': command not found

Ranjith@Ranjith-PC MINGW32 ~/desktop/gitlab (master)
$ git checkout -b feature-branch
Switched to a new branch 'feature-branch'

Ranjith@Ranjith-PC MINGW32 ~/desktop/gitlab (feature-branch)
$ git add raj.txt

Ranjith@Ranjith-PC MINGW32 ~/desktop/gitlab (feature-branch)
$ git status
On branch feature-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   raj.txt


Ranjith@Ranjith-PC MINGW32 ~/desktop/gitlab (feature-branch)
$ git commit -m "Added a new file"
[feature-branch 08259d3] Added a new file
 1 file changed, 1 insertion(+)
 create mode 100644 raj.txt

Ranjith@Ranjith-PC MINGW32 ~/desktop/gitlab (feature-branch)
$ git checkout master
Switched to branch 'master'

Ranjith@Ranjith-PC MINGW32 ~/desktop/gitlab (master)
$ git merge feature-branch
Updating 23dfc4a..08259d3
Fast-forward
 raj.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 raj.txt
```

Experiment 3.

Creating and Managing Branches:

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

Solution:
To stash your changes, switch branches, and then apply the stashed changes in Git, you can use the following commands:

1. Stash your changes:

$ git stash save "Your stash message"

This command will save your changes in a stash, which acts like a temporary storage for changes that are not ready to be committed.

1. Switch to the desired branch:

$ git checkout target-branch

Replace "target-branch" with the name of the branch you want to switch to.

To reapply the stashed changes, switch back to your original branch and execute:

$ git stash pop

If you want to remove the stash after applying it, you can use git stash pop instead of git stash apply.
Remember to replace "Your stash message" and "target-branch" with the actual message you want for your stash and the name of the branch you want to switch to.

You can also save stash by a name using
$Git stash push –m “Say-my-name”

To see the lists of Stash
$Git stash List

Solution

```
Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git status
On branch master
nothing to commit, working tree clean

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   how.py

no changes added to commit (use "git add" and/or "git commit -a")

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git stash save "Added a city"
Saved working directory and index state On master: Added a city

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git checkout master
Already on 'master'

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git checkout featurebranch
Switched to branch 'featurebranch'

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (featurebranch)
$ git status
On branch featurebranch
nothing to commit, working tree clean

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (featurebranch)
$ git checkout master
Switched to branch 'master'

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git stash pop
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   how.py

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (55096684701bd3bf33a69e325489c5e483772ec8)
```

```
Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git stash show
 how.py | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git stash push -m "Say-my-name"
Saved working directory and index state On master: Say-my-name

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git stash show list
error: list is not a valid reference

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git stash list
stash@{0}: On master: Say-my-name
stash@{1}: On master: Added 4th line
```

Experiment 4.

Collaboration and Remote Repositories: How

to create github Account:

https://www.youtube.com/watch?v=QUtk-Uuq9nE

Clone a remote Git repository to your local machine. Solution:
To clone a remote Git repository to your local machine, follow these steps:

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to clone the remote Git repository. You can use the cd command to change your working directory.
3. Use the git clone command to clone the remote repository. Replace <repository_url> with the URL of the remote Git repository you want to clone. For example, if you were cloning a repository from GitHub, the URL might look like this:

$ git clone <repository_url> Here's

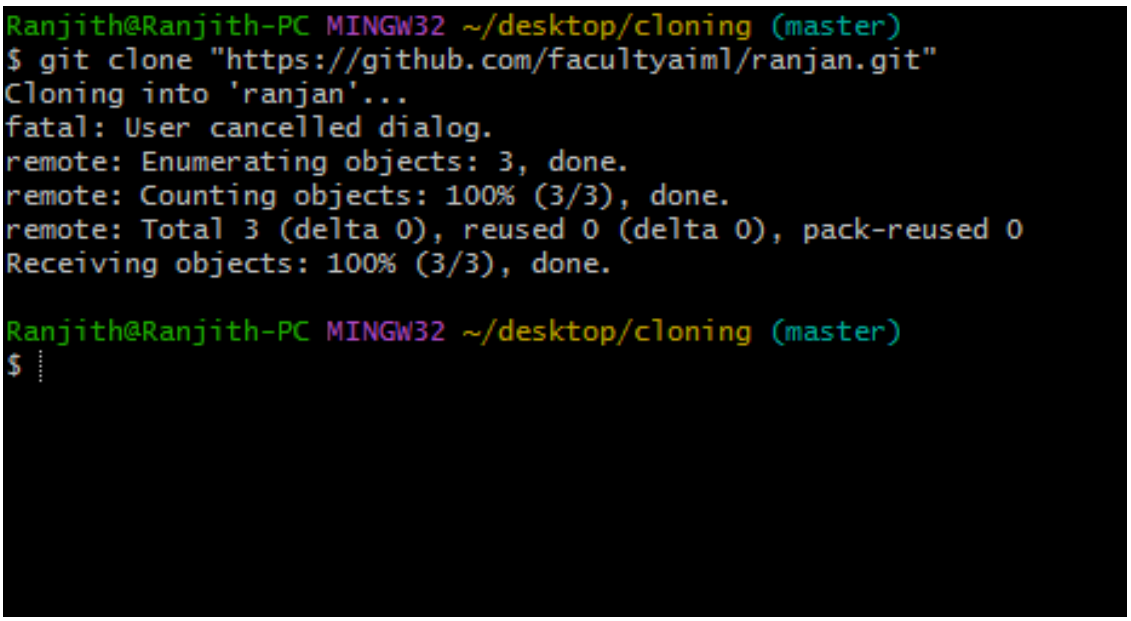
a full example:

$ git clone https://github.com/username/repo-name.git

Replace https://github.com/username/repo-name.git with the actual URL of the repository you want to clone.

1. Git will clone the repository to your local machine. Once the process is complete, you will have a local copy of the remote repository in your chosen directory.

You can now work with the cloned repository on your local machine, make changes, and push those changes back to the remote repository as needed.



Experiment 5.

Git Tag

Git tags are used to label certain commit objects which are major milestones in project development phase like deployment of project in various environment, release codes, bug fixes etc

Lightweight tag

Acts as a pointer to a particular commit.

Creating a lightweight tag:

1. **Get the commit history: git log**
2. **Create a lightweight tag: git tag v1.0 <commit ID>**
3. **Get the details of the tag: git show v1.0**
4. **Similarly, create another tag for some other commit: git tag v2.0 <commit ID>**
5. **Get the names of all the tags created using tag command: git tag**

```
Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git tag v1.0 605422d4a661d911a1da78f1cc2e98303c39a77b

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git show v1.0
commit 605422d4a661d911a1da78f1cc2e98303c39a77b (HEAD -> master, tag: v1.0)
Author: RANJAN V <ranjan@jnnce.ac.in>
Date:   Fri Jan 12 20:15:28 2024 +0530

    added a new comment line

diff --git a/how.py b/how.py
index cf219e7..b4ee3b2 100644
--- a/how.py
+++ b/how.py
@@ -1 +1,2 @@
-printf("Hello world");
\ No newline at end of file
+printf("Hello world");
+//comment line
\ No newline at end of file

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git tag v2.0 4351392fe78bfff264b14e18cb8494c352465388

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git show v2.0
commit 4351392fe78bfff264b14e18cb8494c352465388 (tag: v2.0)
Author: RANJAN V <ranjan@jnnce.ac.in>
Date:   Fri Jan 12 19:30:16 2024 +0530

    Added a file

diff --git a/how.py b/how.py
new file mode 100644
index 0000000..cf219e7
--- /dev/null
+++ b/how.py
@@ -0,0 +1 @@
+printf("Hello world");
\ No newline at end of file

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git tag
v1.0
v2.0
```

Experiment 6

Exploring Git Cherry Pick

You can "cherry pick" any existing commit on another branch into your current branch using git cherry-pick

1. **Checkout existing branch *master* using *git checkout master***
2. **Add a new file called "sample_3.txt" and commit this file to the branch**
3. **Use *git log* to explore the commit id that you would be applying to your custom feature branch**
4. **One you have the commit you want to cherry pick , copy the first few chars in the commit hash**
5. **Execute**
    1. **git cherry-pick <commit-hash>**
    2. **Git add .**

```
Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git commit -m "added a new line for cherrypicking"
[master 73cee1c] added a new line for cherrypicking
 1 file changed, 1 insertion(+), 1 deletion(-)

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git log
commit 73cee1c81b140dbac87f46fc6f256d5f324cdd84 (HEAD -> master)
Author: RANJAN V <ranjan@jnnce.ac.in>
Date:   Fri Jan 12 20:40:25 2024 +0530

    added a new line for cherrypicking

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git checkout featurebranch
Switched to branch 'featurebranch'

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (featurebranch)
$ git checkout featurebranch
Already on 'featurebranch'

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (featurebranch)
$ git cherry-pick 73cee1c
[featurebranch 0d1adda] added a new line for cherrypicking
 Date: Fri Jan 12 20:40:25 2024 +0530
 1 file changed, 1 insertion(+), 1 deletion(-)

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (featurebranch)
$ git log -1
commit 0d1adda0315653f70248e1d7cf5951fa5c53706b (HEAD -> featurebranch)
Author: RANJAN V <ranjan@jnnce.ac.in>
Date:   Fri Jan 12 20:40:25 2024 +0530

    added a new line for cherrypicking

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (featurebranch)
$ git checkout master
Switched to branch 'master'

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git log -1
commit 73cee1c81b140dbac87f46fc6f256d5f324cdd84 (HEAD -> master)
Author: RANJAN V <ranjan@jnnce.ac.in>
Date:   Fri Jan 12 20:40:25 2024 +0530

    added a new line for cherrypicking
```

Experiment 7

Create a account in Github and create your own repository.
Pushing changes to remote repository

1. **Go to your local repository where you have been playing with git so far**
2. **Execute**
    1. ***git remote add origin <repository-link>***
    2. ***git push –f origin master***
3. **As so as you execute push command you will be prompted to login on a separate window , do that and you can see that you changes are pushed to the remote repository**

```
Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git remote add origin https://github.com/facultyaiml/jnnce.git

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git push -u origin main
error: src refspec main does not match any
error: failed to push some refs to 'https://github.com/facultyaiml/jnnce.git'

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git push -u origin main
error: src refspec main does not match any
error: failed to push some refs to 'https://github.com/facultyaiml/jnnce.git'

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ ^C

Ranjith@Ranjith-PC MINGW32 ~/desktop/aiml (master)
$ git push -f origin master
Enumerating objects: 21, done.
Counting objects: 100% (21/21), done.
Delta compression using up to 4 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (21/21), 1.83 KiB | 469.00 KiB/s, done.
Total 21 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/facultyaiml/jnnce/pull/new/master
remote:
To https://github.com/facultyaiml/jnnce.git
 * [new branch]      master -> master
```

Experiment 8.
Analyzing and Changing Git History

Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."

Solution:

To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31" in Git, you can use the git log command with the --author and --since and --until options. Here's the command:

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

This command will display a list of commits made by the author "JohnDoe" that fall within the specified date range, from January 1, 2023, to December 31, 2023. Make sure to adjust the author name and date range as needed for your specific use case.

Solution:

$ git log --author="RANJAN V" --since="2024-01-06" --until="2024-02-06"

Experiment 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?

Solution:

To view the details of a specific commit, including the author, date, and commit message, you can use the git show or git log command with the commit ID. Here are both options:

1. Using git show:

In bash

git show <commit-ID>

Replace <commit-ID> with the actual commit ID you want to view. This command will display detailed information about the specified commit, including the commit message, author, date, and the changes introduced by that commit.

For example:

$ git show abc123

2. Using git log:

$ git log -n 1 <commit-ID>

The -n 1 option tells Git to show only one commit. Replace <commit-ID> with the actual commit ID. This command will display a condensed view of the specified commit, including its commit message, author, date, and commit ID.

```
Ranjith@Ranjith-PC MINGW32 ~/desktop/atc (master)
$ git show 57a3e
commit 57a3e62c6db0e5cecc5c497c35a861a34f5b1ee5 (HEAD -> master, featurebranch)
Author: RANJAN V <ranjan@jnnce.ac.in>
Date:   Tue Feb 6 11:43:57 2024 +0530

    For cherrypicking

diff --git a/hww.txt b/hww.txt
new file mode 100644
index 0000000..8395e98
--- /dev/null
+++ b/hww.txt
@@ -0,0 +1 @@
+For cherrypicking
\ No newline at end of file

Ranjith@Ranjith-PC MINGW32 ~/desktop/atc (master)
$ git log -n 1 57a3e
commit 57a3e62c6db0e5cecc5c497c35a861a34f5b1ee5 (HEAD -> master, featurebranch)
Author: RANJAN V <ranjan@jnnce.ac.in>
Date:   Tue Feb 6 11:43:57 2024 +0530

    For cherrypicking
```

Experiment 10.

Analyzing and Changing Git History

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

Solution:

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.

Once you give the command revert it will ask you for the type the message in new window type the message for reverting and after the message

Type ctrl C

And Type :qa!

ENTER

Which will delete the specific file you committed

```
Ranjith@Ranjith-PC MINGW32 ~/desktop/atc (master)
$ git show
commit 57a3e62c6db0e5cecc5c497c35a861a34f5b1ee5 (HEAD -> master, featurebranch)
Author: RANJAN V <ranjan@jnnce.ac.in>
Date:   Tue Feb 6 11:43:57 2024 +0530

    For cherrypicking

diff --git a/hww.txt b/hww.txt
new file mode 100644
index 0000000..8395e98
--- /dev/null
+++ b/hww.txt
@@ -0,0 +1 @@
+For cherrypicking
\ No newline at end of file

Ranjith@Ranjith-PC MINGW32 ~/desktop/atc (master)
$ git revert 57a3e62
[master d8952d5] Revert "For cherrypicking"
 1 file changed, 1 deletion(-)
 delete mode 100644 hww.txt
```

Experiment 11.

Collaboration and Remote Repositories:

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

Solution:

To fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch in Git, follow these steps:

1. Open your terminal or command prompt.
2. Make sure you are in the local branch that you want to rebase. You can switch to the branch using the following command, replacing <branch-name> with your actual branch name:

$ git checkout <branch-name>

1. Fetch the latest changes from the remote repository. This will update your local repository with the changes from the remote without merging them into your local branch:

$ git fetch origin

Here, origin is the default name for the remote repository. If you have multiple remotes, replace origin with the name of the specific remote you want to fetch from.

1. Once you have fetched the latest changes, rebase your local branch onto the updated remote branch:

$ git rebase origin/<branch-name>

Replace <branch-name> with the name of the remote branch you want to rebase onto. This command will reapply your local commits on top of the latest changes from the remote branch, effectively incorporating the remote changes into your branch history.

Resolve any conflicts that may arise during the rebase process. Git will stop and notify you if there are conflicts that need to be resolved. Use a text editor to edit the conflicting files, save the changes, and then continue the rebase with:

$ git rebase --continue

- After resolving any conflicts and completing the rebase, you have successfully updated your local branch with the latest changes from the remote branch.
- If you want to push your rebased changes to the remote repository, use the git push command. However, be cautious when pushing to a shared remote branch, as it can potentially overwrite other developers' changes:

$ git push origin <branch-name>

Replace <branch-name> with the name of your local branch. By following these steps, you can keep your local branch up to date with the latest changes from the remote repository and maintain a clean and linear history through rebasing.

Experiment 12.

Analysing and Changing Git History

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

Solution:

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

This command will show the last five commits in the repository's history. You can adjust the number after -n to display a different number of commits if needed.