Date: 15-05-2025

1. What is the role of data engineer?
    - A **Data Engineer** builds and maintains data pipelines, ensuring reliable, clean, and accessible data for analysis and business use.
2. What is Big Data?
    - **Big Data** refers to extremely large and complex data sets that traditional data processing tools cannot handle efficiently.
3. How do you classify the given data is Big Data?
    - **Volume** – The vast amount of data generated every second.
    - **Velocity** – The speed at which data is generated and processed.
    - **Variety** – Different types of data (structured, unstructured, semi-structured).
    - **Veracity** – The accuracy and trustworthiness of the data.
    - **Value** – The usefulness of the data in making decisions.
4. How to handle this Big Data?
    - **Distributed Storage** – Use systems like **HDFS**, **Amazon S3**, or **Azure Blob** to store large datasets across multiple nodes.
    - **Parallel Processing** – Tools like **Apache Hadoop**, **Apache Spark**, and **Flink** process data in parallel to speed up computation.
    - **Scalable Databases** – Use NoSQL databases like **MongoDB**, **Cassandra**, or **HBase** for flexible and scalable data management.
5. What is Git and GitHub?
    - **Git** is a distributed version control system used to track changes in source code during software development.
    - **GitHub** is a cloud-based platform that hosts Git repositories and helps developers collaborate using Git.
6. Basic commands for Command Prompt
    - Create a file:- echo. > filename.txt
    - Edit a file:- notepad filename.txt
    - Display contents of a file:- type filename.txt
    - Rename a file:- rename oldname.txt newname.txt
    - Delete a file:- del filename.txt

Git Commands

1. Git Clone:

It is used to create a local copy of a remote repository. This command not only copies all the files, but also brings the entire Git history of the project.

git clone <repository-url>

```
C:\Users\DivakarC\Desktop\Report demo>git init
Initialized empty Git repository in C:/Users/DivakarC/Desktop/Report demo/.git/

C:\Users\DivakarC\Desktop\Report demo>git clone https://github.com/divakar-bytewise/Practice-repository.git
Cloning into 'Practice-repository'...
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.
```

2. Git Config:

git config is used to **set Git configuration values**. It controls aspects of Git's behaviour and environment. We can use it to set user information, editor preferences, aliases, and much more.

Setting User Identity:

1. These are **mandatory** for commits:

   git config --global user.name " Name"

   git config --global user.email " .email@example.com"

```
C:\Users\DivakarC\Desktop>git config --global user.name "diakar-bytewise"

C:\Users\DivakarC\Desktop>git config --global user.email "divakar.c@diggibyte.com"
```

2. View Configurations:

   git config --global –list

   git config user.name

```
C:\Users\DivakarC\Desktop>git config --global --list
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=diakar-bytewise
user.email=divakar.c@diggibyte.com
core.editor=notepad
```

```
C:\Users\DivakarC\Desktop>git config user.name
diakar-bytewise
```

3. Creating Git Aliases:

   git config --global alias.co checkout

   git config --global alias.st status

   git config --global alias.br branch

   git config --global alias.cm "commit -m"

```
C:\Users\DivakarC\Desktop>git config --global alias.co checkout

C:\Users\DivakarC\Desktop>git config --global alias.st status

C:\Users\DivakarC\Desktop>git config --global alias.br branch

C:\Users\DivakarC\Desktop>git config --global alias.cm "commit -m"
```

Date: 16-05-2025

1. Git add:

Git command used to add changes in the working directory to the staging area (index). This prepares the changes to be committed in your Git repository.

Commands and Snapshots:

1. *git add .*

```
C:\Users\DivakarC\Desktop\Practice-repository>git add .

C:\Users\DivakarC\Desktop\Practice-repository>git commit -m "Added a text file"
[main ca3a218] Added a text file
 1 file changed, 1 insertion(+)
 create mode 100644 text.txt
```

2. *git add file_name*

```
C:\Users\DivakarC\Desktop\Practice-repository>git add text.txt

C:\Users\DivakarC\Desktop\Practice-repository>git commit -m "Text file updated"
[main c08a1ba] Text file updated
 1 file changed, 1 insertion(+), 1 deletion(-)
```

3. *git add **

```
C:\Users\DivakarC\Desktop\Practice-repository>git add *.txt

C:\Users\DivakarC\Desktop\Practice-repository>git commit -m "Text file updated"
[main 82c8c03] Text file updated
 1 file changed, 1 insertion(+), 1 deletion(-)
```

4. *git add foldername*

```
C:\Users\DivakarC\Desktop\Practice-repository>git add Feature/

C:\Users\DivakarC\Desktop\Practice-repository>git commit -m "Added a folder"
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

C:\Users\DivakarC\Desktop\Practice-repository>git push origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 814 bytes | 407.00 KiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/divakar-bytewise/Practice-repository.git
   4e252f1..82c8c03  main -> main
```

2. Git push:

git push is a command used to upload our local repository content to a remote repository.

git push <remote-name> <branch-name>

```
C:\Users\DivakarC\Desktop\Practice-repository>git push origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 814 bytes | 407.00 KiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/divakar-bytewise/Practice-repository.git
   4e252f1..82c8c03  main -> main
```

3. Git pull:

Git command that downloads content from a remote repository and integrates it into our current branch.

- **Fetches** the latest updates from the remote repository.
- **Merges** those updates into the current local branch.

git pull <remote-name> <branch-name>

```
C:\Users\DivakarC\Desktop\Practice-repository>git pull origin main
From https://github.com/divakar-bytewise/Practice-repository
 * branch            main       -> FETCH_HEAD
Already up to date.
```

4. Git fetch:

git fetch is a command that retrieves the latest updates from a remote repository but does **not** integrate them into your working files or current branch.

git fetch <remote> <branch>

```
C:\Users\DivakarC\Desktop\Practice-repository>git fetch origin main
From https://github.com/divakar-bytewise/Practice-repository
 * branch            main       -> FETCH_HEAD
```

5. Git merge:

git merge integrates changes from one branch into the current branch**.**

git merge <branch-name>

```
C:\Users\DivakarC\Desktop\Practice-repository>git merge origin main
Already up to date.
```

6. Git rebase:

git rebase is used to **reapply commits** on top of another base tip.

git rebase <base-branch>

```
C:\Users\DivakarC\Desktop\Practice-repository>git rebase main
Current branch feature is up to date.
```

7. Git revert:

git revert is a command used to undo the effects of a specific commit by creating a new commit that reverses the changes made in the original commit.

git revert <commit-hash>

```
C:\Users\DivakarC\Desktop\Practice-repository>git log --oneline
1c55a99 (HEAD -> feature) "working on revert"
d20bf97 (origin/feature) update
5f05e2f Revert "Text file updated"
82c8c03 (origin/main, origin/HEAD, main) Text file updated
c08a1ba Text file updated
ca3a218 Added a text file
4e252f1 Initial commit
```

8. Git cherry-pick:

git cherry-pick is a command that applies the changes introduced by an existing commit to the current working branch, regardless of its origin.

git cherry-pick <commit-hash>

```
C:\Users\DivakarC\Desktop\Practice-repository>git cherry-pick 1c55a99
Auto-merging text.txt
[main d880ab4] "working on revert"
 Date: Fri May 23 18:41:38 2025 +0530
 1 file changed, 2 insertions(+), 1 deletion(-)

C:\Users\DivakarC\Desktop\Practice-repository>git log --oneline
d880ab4 (HEAD -> main) "working on revert"
9ee02c1 (origin/main, origin/HEAD) Cheery-pick update
6972b19 "working on revert"
82c8c03 Text file updated
c08a1ba Text file updated
ca3a218 Added a text file
4e252f1 Initial commit
```

9. Git reset:

git reset is one of the most powerful and potentially destructive commands in Git. It is used to undo changes by resetting the current HEAD to a specified state.

git reset [<mode>] [<commit>]

git reset --hard will **erase** uncommitted changes. Always double-check before using it.

- Undo the last commit but keep changes staged:

  git reset --soft HEAD~1

- Unstage changes without deleting them:

  git reset HEAD <file>

```
C:\Users\DivakarC\Desktop\Practice-repository>git reset --hard d880ab4
HEAD is now at d880ab4 "working on revert"
```

Date: 19-05-2025

1. Branching and Merging:

Branching allows you to diverge from the main line of development and work independently on features, fixes, or experiments.

Merging integrates the changes from one branch into another.

Commands and Snapshots:

1. *git branch*

```
C:\Users\DivakarC\Desktop\Practice-repository>git branch
  feature
* main
```

2. *git branch <branch-name>*

```
C:\Users\DivakarC\Desktop\Practice-repository>git branch feature-branch

C:\Users\DivakarC\Desktop\Practice-repository>git branch
  feature
  feature-branch
* main
```

3. *git checkout <branch-name>*

```
C:\Users\DivakarC\Desktop\Practice-repository>git checkout feature-branch
Switched to branch 'feature-branch'

C:\Users\DivakarC\Desktop\Practice-repository>git branch
  feature
* feature-branch
  main
```

4. *git switch <branch-name>*

```
C:\Users\DivakarC\Desktop\Practice-repository>git switch main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

5. git checkout -b <branch-name>

```
C:\Users\DivakarC\Desktop\Practice-repository>git checkout -b new-branch
Switched to a new branch 'new-branch'
```

6. git merge <branch-name>

```
C:\Users\DivakarC\Desktop\Practice-repository>git merge main
Already up to date.
```

7. git log --oneline --graph --all

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git log --oneline --graph --all
*   d9225b0 (HEAD -> main, origin/main, origin/HEAD) Merge branch 'main' of https://github.com/divakar-bytewise/Practice-repo
|\
| *   e0b2919 Merge pull request #2 from divakar-bytewise/feature
| |\
| | *   769fdb0 (origin/feature, feature) Merge branch 'feature' of https://github.com/divakar-bytewise/Practice-repo into feature
| | |\
| | | *   66c2068 Merge branch 'feature' of https://github.com/divakar-bytewise/Practice-repo into feature
| | | |\
| | * | | 57fce04 Hii
| | * | | 59752ff Only thye textfile is ignored
| | * | | fc924b8 Remove .txt files from tracking
| | |/ /
| | * | 75f90e1 working on edit
| | * | 3f3fba1 git ignore file
| | * | ce2cc69 git ignore file
| * | |   d7f9ae8 Merge branch 'main' of https://github.com/divakar-bytewise/Practice-repo
| |\ \ \
| | * | | 5ad1f1d Working with stash, relog and cherry-pick
| * | | | 9fa2f55 Working with stash, relog and cherry-pick
| * | | | f6283a0 Combined the commit
* | | | | b881583 Working with stash, relog and cherry-pick
* | | | | 283049e Working with stash, relog and cherry-pick
* | | | | acbf444 Combined the commit
* | | | | a23e666 Working with reword
* | | | | d50daf1 Git reword in IR
* | | | | 767651d reword b5adf8c re-word
| |/ / /
|/| | |
| | | | * ee8542f (origin/recover-branch, recover-branch) Working with stash
| |_|_|/
```

2. Collaborative work:

Collaborative development involves **multiple people working on the same codebase**, often using **branches, remotes, pull requests, and conflict resolution**.

1. Clone a Repository:

   git clone <repo-url>
2. Creating and Switch to a New Branch:

   git checkout -b feature
3. Add and Commit Changes:

   git add .

   git commit
4. Create a Pull Request (PR)

   - Go to your GitHub repo
   - Click **"Compare & Pull Request"**
   - Add a title and description

- Submit for review

5. Pull Latest Changes from Main:
    - *git checkout main*
    - *git pull origin main*
6. Conflict Handling:
   <<<<<<< HEAD

   your code

   =======

   someone else's code

   >>>>>>> branch-name

    - *git add .*
    - *git rebase –continue*

3. Git ignore:

The .gitignore file tells Git which files or directories to ignore

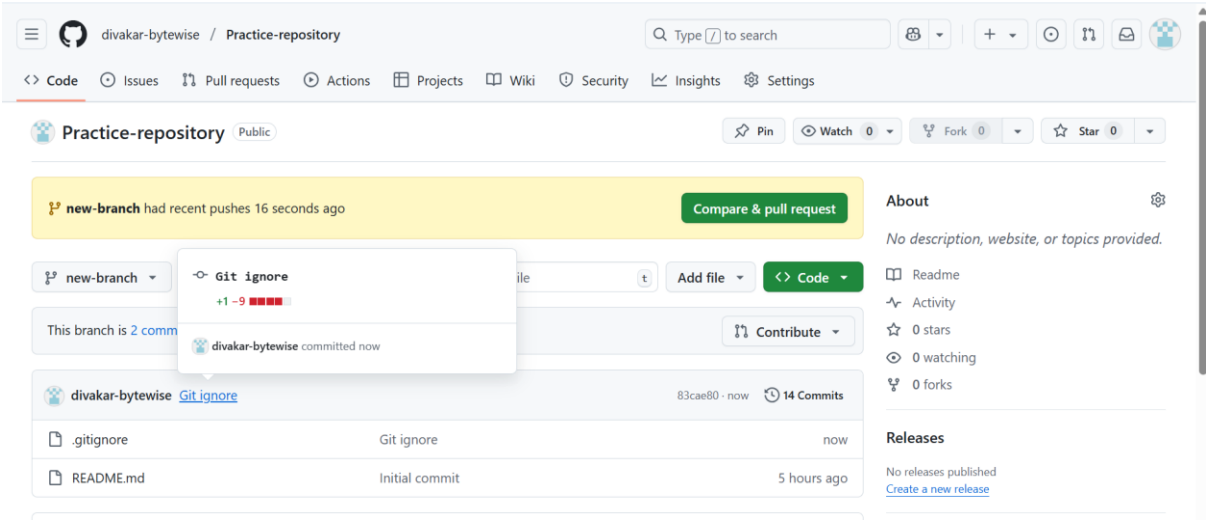
Commands and Snapshots:

1. Creating a new file:
   .gitignore

```
C:\Users\DivakarC\Desktop\Practice-repository>git add .

C:\Users\DivakarC\Desktop\Practice-repository>git commit
hint: Waiting for your editor to close the file... unix2dos: converting file C:/Users/DivakarC/Desktop/Practice-reposito
ry/.git/COMMIT_EDITMSG to DOS format...
dos2unix: converting file C:/Users/DivakarC/Desktop/Practice-repository/.git/COMMIT_EDITMSG to Unix format...
Aborting commit due to empty commit message.

C:\Users\DivakarC\Desktop\Practice-repository>git commit -m "Git ignore"
[new-branch bbb83cb] Git ignore
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
```

2. If Git is still tracking the file:

   git rm --cached filename



4. Git Tagging:

Tags are like bookmarks that point to specific commits in your repo. Most commonly used to mark release versions.

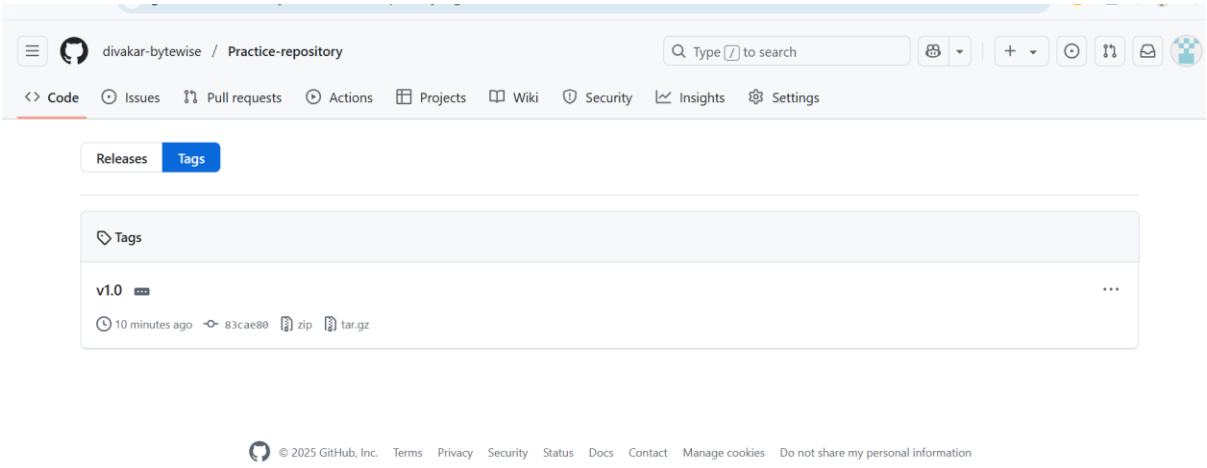
Commands and Snapshots:

- *git tag v1.0*
- *git tag -a v1.0 -m "Version 1.0 Release”*
- *git log –oneline*
- *git tag -a v2.0 bbb83cb -m "Version 2.0 Release"*
- *git tag*
- *git tag -d v2.0*

```
C:\Users\DivakarC\Desktop\Practice-repository>git tag v1.0
```

```
C:\Users\DivakarC\Desktop\Practice-repository>git log --oneline
83cae80 (HEAD -> new-branch, tag: v1.0, origin/new-branch) Git ignore
bbb83cb Git ignore
3690774 (origin/main, origin/HEAD, main, feature-branch) Merge branch 'main' of https://github.com/divakar-bytewise/Prac
tice-repository
d880ab4 "working on revert"
e350fa3 Merge pull request #1 from divakar-bytewise/feature
0656b17 Merge branch 'main' into feature
9ee02c1 Cheery-pick update
6972b19 "working on revert"
d20bf97 (origin/feature) update
5f05e2f Revert "Text file updated"
82c8c03 Text file updated
c08a1ba Text file updated
ca3a218 Added a text file
4e252f1 Initial commit
```

```
C:\Users\DivakarC\Desktop\Practice-repository>git push origin v1.0
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/divakar-bytewise/Practice-repository.git
 * [new tag]         v1.0 -> v1.0
```



Date: 20-05-2025

1. **Workflow:**

1. Creating a Local Git Repository
    - *git init → Initialize a Git repo*
2. Creating & Committing Files
    - *echo "text" > file.txt → Create a file*
    - *git add file.txt → Stage the file*
    - *git commit -m "Initial commit" → Save the snapshot*
3. Viewing Commit History
    - *git log → See previous commits*
    - *git status → View current state of working directory*
4. Modifying Files
    - *Make edits to the file manually*
    - *git status → See "modified" file*
    - *git add . → Stage the changes*
    - *git commit -m "Updated file" → Save changes*

5. Remote Setup (GitHub)
    - *git remote add origin <repo-url> → Connect to GitHub*
    - *git push -u origin main → Push to GitHub*
6. Cloning the Repo
    - *git clone <repo-url> → Make a copy locally*
7. Fetching & Pulling Changes
    - *git fetch → Get latest changes from remote*
    - *git pull → Download & merge the changes*

2. Git Config:

git config is used to **set Git configuration values**. It controls aspects of Git's behaviour and environment. We can use it to set user information, editor preferences, aliases, and much more.

Setting User Identity:

1. These are **mandatory** for commits:
   git config --global user.name " Name"
   git config --global user.email " .email@example.com"

```
C:\Users\DivakarC\Desktop>git config --global user.name "diakar-bytewise"

C:\Users\DivakarC\Desktop>git config --global user.email "divakar.c@diggibyte.com"
```

2. View Configurations:
   git config --global –list
   git config user.name

```
C:\Users\DivakarC\Desktop>git config --global --list
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=diakar-bytewise
user.email=divakar.c@diggibyte.com
core.editor=notepad
```

```
C:\Users\DivakarC\Desktop>git config user.name
diakar-bytewise
```

3. Creating Git Aliases:

   git config --global alias.co checkout

   git config --global alias.st status

   git config --global alias.br branch

   git config --global alias.cm "commit -m"

```
C:\Users\DivakarC\Desktop>git config --global alias.co checkout

C:\Users\DivakarC\Desktop>git config --global alias.st status

C:\Users\DivakarC\Desktop>git config --global alias.br branch

C:\Users\DivakarC\Desktop>git config --global alias.cm "commit -m"
```

3. Undoing Changes:

1. Git revert:

git revert is a command used to undo the effects of a specific commit by creating a new commit that reverses the changes made in the original commit.

git revert <commit-hash>

```
C:\Users\DivakarC\Desktop\Practice-repository>git log --oneline
1c55a99 (HEAD -> feature) "working on revert"
d20bf97 (origin/feature) update
5f05e2f Revert "Text file updated"
82c8c03 (origin/main, origin/HEAD, main) Text file updated
c08a1ba Text file updated
ca3a218 Added a text file
4e252f1 Initial commit
```

2. Git reset:

git reset is one of the most powerful and potentially destructive commands in Git. It is used to undo changes by resetting the current HEAD to a specified state.

git reset [<mode>] [<commit>]

git reset --hard will **erase** uncommitted changes. Always double-check before using it.

- Undo the last commit but keep changes staged:

  git reset --soft HEAD~1

- Unstage changes without deleting them:

  git reset HEAD <file>

```
C:\Users\DivakarC\Desktop\Practice-repository>git reset --hard d880ab4
HEAD is now at d880ab4 "working on revert"
```

4. Git rebase:

git rebase is used to **reapply commits** on top of another base tip.

git rebase <base-branch>

```
C:\Users\DivakarC\Desktop\Practice-repository>git rebase main
Current branch feature is up to date.
```

5. Git Cherry-pick:

git cherry-pick is a command that applies the changes introduced by an existing commit to the current working branch, regardless of its origin.

git cherry-pick <commit-hash>

6.

```
C:\Users\DivakarC\Desktop\Practice-repository>git cherry-pick 1c55a99
Auto-merging text.txt
[main d880ab4] "working on revert"
 Date: Fri May 23 18:41:38 2025 +0530
 1 file changed, 2 insertions(+), 1 deletion(-)

C:\Users\DivakarC\Desktop\Practice-repository>git log --oneline
d880ab4 (HEAD -> main) "working on revert"
9ee02c1 (origin/main, origin/HEAD) Cheery-pick update
6972b19 "working on revert"
82c8c03 Text file updated
c08a1ba Text file updated
ca3a218 Added a text file
4e252f1 Initial commit
```

Date: 21-05-2025

1. Git Hooks:

Git Hooks are scripts that Git automatically runs before or after certain events like commits, merges, and pushes to automate tasks.

Types of Git Hooks:

1. **Client-side hooks:**

   Go to your repo's hooks folder:

   cd your-project/.git/hooks

   Rename a sample or create a new file:

   rename pre-commit.sample pre-commit

   Use Notepad:

   notepad pre-commit

```
C:\Users\DivakarC\Desktop\Practice-repository>cd .git/hooks
```

```
C:\Users\DivakarC\Desktop\Practice-repository\.git\hooks>rename pre-commit.sample pre-commit

C:\Users\DivakarC\Desktop\Practice-repository\.git\hooks>notepad pre-commit
```

```
C:\Users\DivakarC\Desktop\Practice-repository\.git\hooks>cd C:\Users\DivakarC\Desktop\Practice-repository

C:\Users\DivakarC\Desktop\Practice-repository>git commit -m "check"
./.git/hooks/pre-commit:2:if grep -rn "TODO" .; then
./.git/hooks/pre-commit:3:  echo "Remove TODOs before committing."
Remove TODOs before committing.
```

2. Git interactive rebase:

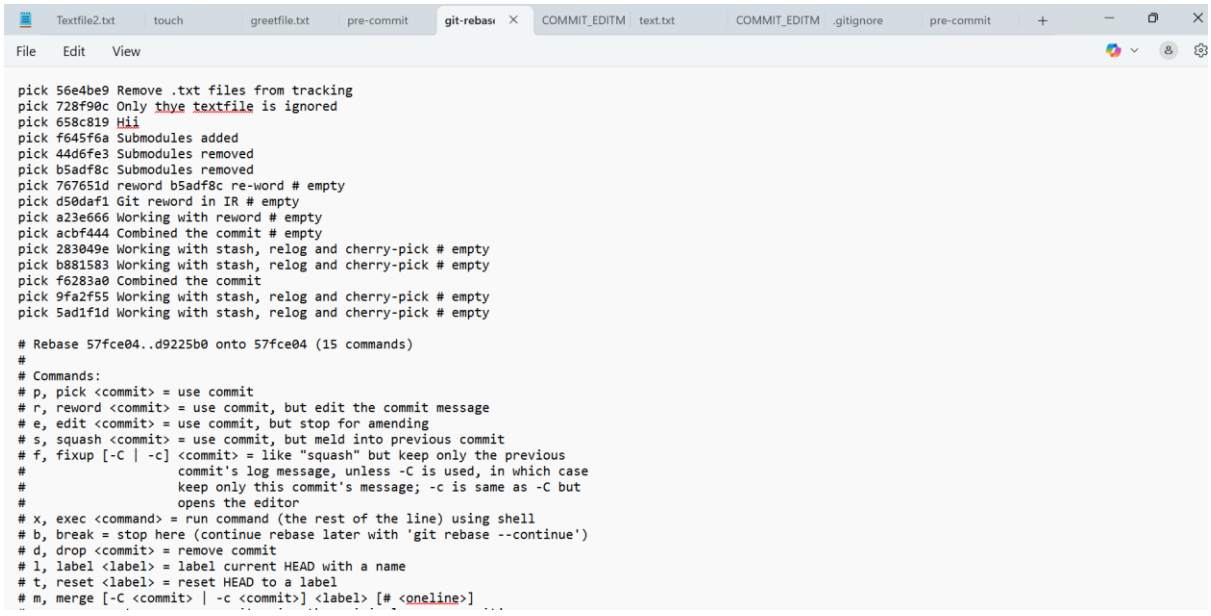
Interactive Rebase is a Git feature that allows you to edit, reorder, squash, or delete commits in a controlled way, providing a clean and organized commit history.

git rebase -i HEAD~<n>

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git rebase -i 767651d
hint: Waiting for your editor to close the file... unix2dos: converting file C:/Users/DivakarC/Desktop/Git-Practice/Practice-repo/.gi
t/rebase-merge/git-rebase-todo to DOS format...
dos2unix: converting file C:/Users/DivakarC/Desktop/Git-Practice/Practice-repo/.git/rebase-merge/git-rebase-todo to Unix format...
Stopped at 767651d...  first edit
You can amend the commit now, with

  git commit --amend

Once you are satisfied with your changes, run

  git rebase --continue
```

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git log --oneline
d9225b0 (HEAD -> main, origin/main, origin/HEAD) Merge branch 'main' of https://github.com/divakar-bytewise/Practice-repo
b881583 Working with stash, relog and cherry-pick
283049e Working with stash, relog and cherry-pick
acbf444 Combined the commit
a23e666 Working with reword
d50daf1 Git reword in IR
767651d reword b5adf8c re-word
e0b2919 Merge pull request #2 from divakar-bytewise/feature
769fdb0 (origin/feature, feature) Merge branch 'feature' of https://github.com/divakar-bytewise/Practice-repo into feature
57fce04 Hii
59752ff Only thye textfile is ignored
fc924b8 Remove .txt files from tracking
66c2068 Merge branch 'feature' of https://github.com/divakar-bytewise/Practice-repo into feature
75f90e1 working on edit
3f3fba1 git ignore file
ce2cc69 git ignore file
d7f9ae8 Merge branch 'main' of https://github.com/divakar-bytewise/Practice-repo
9fa2f55 Working with stash, relog and cherry-pick
f6283a0 Combined the commit
5ad1f1d Working with stash, relog and cherry-pick
b5adf8c Submodules removed
44d6fe3 Submodules removed
f645f6a Submodules added
658c819 Hii
728f90c (tag: v1.0, origin/feature-branch, feature-branch) Only thye textfile is ignored
56e4be9 Remove .txt files from tracking
fa929dc gitignore
03a0242 (upstream/main) Merge branch 'feature-branch' of https://github.com/divakar-bytewise/Practice-repo
7f9ac12 Merge branch 'feature-branch'
7a4ac5b (upstream/feature-branch) Add files via upload
3b4d3e1 add new txt file
5279fe2 add new txt file
```

https://github.com/divakar-bytewise/Practice-repo
Ctrl+Click to follow link



In the opened editor:

- Use pick to keep as-is.
- Use reword to change commit message.
- Use squash to merge commits.
- Use drop to delete a commit.

3. Git GUIs:

Git GUIs are tools that provide a **visual interface** for Git operations instead of using command-line commands, making version control more user-friendly—especially for beginners.

1. **GitHub Desktop** (Windows, macOS):
   Offers easy integration with GitHub. It simplifies tasks like branching, committing, and creating pull requests with a user-friendly interface.
2. **Sourcetree** (Windows, macOS):
   Provides a visual representation of branches, helps resolve merge conflicts, and supports stash management for handling temporary changes.
3. **GitKraken** (Windows, macOS, Linux):A modern, cross-platform Git GUI with an intuitive interface. It includes features like Glo boards and team collaboration tools.
4. **TortoiseGit** (Windows):
   Integrates Git directly into Windows File Explorer, making it easy to use Git operations from the context menu.
5. **VS Code Git GUI** (Cross-platform):
   Comes built-in with Visual Studio Code. It offers a source control view that supports committing, branching, and merging changes through an easy-to-use interface.

Basic Actions in Git GUI Tools:

- Clone a repository – Choose a repo URL and destination.
- Stage and commit changes – View modified files and commit with messages.
- Push and pull – Sync local and remote repositories.
- Merge branches – Handle branches with visual merge tools.
- Resolve conflicts – Use visual diff/merge editors.

Date: 22-05-2025

1. Submodules in Git

- Submodules allow you to keep a Git repository as a subdirectory of another repository.

Commands and Snapshots:

1. Add a Submodule:

   git submodule add <repository-url> <destination-folder>

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git submodule add https://github.com/divakar-bytewise/Assignment-1-
DBT-.git
Cloning into 'C:/Users/DivakarC/Desktop/Git-Practice/Practice-repo/Assignment-1-DBT-'...
remote: Enumerating objects: 31, done.
remote: Counting objects: 100% (31/31), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 31 (delta 7), reused 23 (delta 4), pack-reused 0 (from 0)
Receiving objects: 100% (31/31), 1.70 MiB | 7.73 MiB/s, done.
Resolving deltas: 100% (7/7), done.
warning: in the working copy of '.gitmodules', LF will be replaced by CRLF the next time Git touches it
```

2. Initialize Submodules:

   Initializes the submodules listed in .gitmodules. *git submodule init*

3. Clone a Repo with Submodules:

   git clone --recurse-submodules <repository-url>

```
Cloning into 'C:/Users/DivakarC/Desktop/Git-Practice/Practice-repo/Assignment-1-DBT-'...
remote: Enumerating objects: 31, done.
remote: Counting objects: 100% (31/31), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 31 (delta 7), reused 23 (delta 4), pack-reused 0 (from 0)
Receiving objects: 100% (31/31), 1.70 MiB | 7.73 MiB/s, done.
Resolving deltas: 100% (7/7), done.
warning: in the working copy of '.gitmodules', LF will be replaced by CRLF the next time Git touches it
```

4. Fetch Latest Changes in Submodule:

   Pulls the latest commit from the submodule's default branch *git submodule update –remote*

5. Check Submodule Status:

   git submodule status

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git submodule status
 cb79de4c85dd31c16b77ce47387668980f38c292 Assignment-1-DBT- (heads/main)
```

6. Remove a Submodule:

   git rm -f <path-to-submodule>

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git rm -f C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo\Assi
gnment-1-DBT-
rm 'Assignment-1-DBT-'
```

7. When others clone your repo:

   To automatically clone both the main repo and all its submodules *git clone --recurse-submodules <repo>*

2. Stashing in Git

- git stash temporarily saves your *uncommitted changes* (both staged and unstaged) so you can work on something else and come back to them later.

Commands and Snapshots:

1. Temporarily saving changes using:

   git stash

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git stash
Saved working directory and index state WIP on main: 44d6fe3 Submodules removed
```

2. Save stash with a custom message:

   git stash save "message"

3. View list of stashed changes:

   git stash list

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git stash list
stash@{0}: WIP on main: 44d6fe3 Submodules removed
```

4. Reapply the most recent stash:

   git stash apply

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git stash apply
On branch main
Your branch is up to date with 'origin/main'.

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:   Textfile2.txt

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

5. Reapply the most recent stash and remove it:

   git stash pop

6. Delete a specific stash:
   git stash drop

7. Delete all stashes:
   git stash clear

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git stash clear
```

3. Advanced Topics in Git

1. Reflog in git:

- Git tracks every movement of HEAD and updates it. Even if you reset, delete a branch, or lose commits, git reflog allows you to see the previous state of your repository and recover from mistakes.

Commands and Snapshots:

1. Syntax:
   git reset --hard b5adf8c, git reflog

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git reset --hard b5adf8c
HEAD is now at b5adf8c Submodules removed

C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git log --oneline
b5adf8c (HEAD -> main) Submodules removed
44d6fe3 Submodules removed
f645f6a Submodules added
658c819 (origin/feature, feature) Hii
728f90c (tag: v1.0, origin/feature-branch, feature-branch) Only thye textfile is ignored
56e4be9 Remove .txt files from tracking
fa929dc gitignore
03a0242 (upstream/main) Merge branch 'feature-branch' of https://github.com/divakar-bytewise/Practice-repo
7f9ac12 Merge branch 'feature-branch'
7a4ac5b (upstream/feature-branch) Add files via upload
3b4d3e1 add new txt file
5279fe2 add new txt file
9808add no
c3d57ca Adding textfile
14d252d Add files via upload
4cb9370 Revert "The new textfile2 is added"
43d4901 The new textfile2 is added
44eb1a8 Add files via upload
5e31fbb Add files via upload
a749e63 Textfile added
5b82297 Initial commit
```

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git reflog
b5adf8c (HEAD -> main) HEAD@{0}: reset: moving to b5adf8c
ee8542f (origin/main, origin/HEAD) HEAD@{1}: reset: moving to ee8542f
ee8542f (origin/main, origin/HEAD) HEAD@{2}: checkout: moving from main to main
ee8542f (origin/main, origin/HEAD) HEAD@{3}: checkout: moving from main to main
ee8542f (origin/main, origin/HEAD) HEAD@{4}: rebase (continue) (finish): returning to refs/heads/main
ee8542f (origin/main, origin/HEAD) HEAD@{5}: commit (rebase): Working with stash
b5adf8c (HEAD -> main) HEAD@{6}: rebase (start): checkout b5adf8c
b5adf8c (HEAD -> main) HEAD@{7}: rebase (continue) (finish): returning to refs/heads/main
b5adf8c (HEAD -> main) HEAD@{8}: rebase (start): checkout b5adf8c
b5adf8c (HEAD -> main) HEAD@{9}: rebase (skip) (finish): returning to refs/heads/main
b5adf8c (HEAD -> main) HEAD@{10}: rebase (start): checkout b5adf8c
b5adf8c (HEAD -> main) HEAD@{11}: rebase (continue) (finish): returning to refs/heads/main
b5adf8c (HEAD -> main) HEAD@{12}: rebase (start): checkout b5adf8c
b5adf8c (HEAD -> main) HEAD@{13}: commit: Submodules removed
44d6fe3 HEAD@{14}: reset: moving to HEAD
44d6fe3 HEAD@{15}: commit: Submodules removed
f645f6a HEAD@{16}: commit: Submodules added
658c819 (origin/feature, feature) HEAD@{17}: rebase (continue) (finish): returning to refs/heads/main
658c819 (origin/feature, feature) HEAD@{18}: rebase (start): checkout 658c8190fa0f8a1df73029c2ab95bc44f8dd4935
658c819 (origin/feature, feature) HEAD@{19}: rebase (abort): updating HEAD
658c819 (origin/feature, feature) HEAD@{20}: rebase (start): checkout 658c8190fa0f8a1df73029c2ab95bc44f8dd4935
658c819 (origin/feature, feature) HEAD@{21}: rebase (finish): returning to refs/heads/main
658c819 (origin/feature, feature) HEAD@{22}: rebase (start): checkout 658c8190fa0f8a1df73029c2ab95bc44f8dd4935
658c819 (origin/feature, feature) HEAD@{23}: merge feature: Fast-forward
728f90c (tag: v1.0, origin/feature-branch, feature-branch) HEAD@{24}: checkout: moving from feature to main
658c819 (origin/feature, feature) HEAD@{25}: commit: Hii
728f90c (tag: v1.0, origin/feature-branch, feature-branch) HEAD@{26}: checkout: moving from main to feature
728f90c (tag: v1.0, origin/feature-branch, feature-branch) HEAD@{27}: checkout: moving from feature to main
```

2. Recovering the commit:

   git cherry-pick ee8542f, git commit --allow-empty

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git cherry-pick ee8542f
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'
On branch main
Your branch is up to date with 'origin/main'.

You are currently cherry-picking commit ee8542f.
  (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

C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git commit --allow-empty
hint: Waiting for your editor to close the file... unix2dos: converting file C:/Users/DivakarC/Desktop/Git-Practice/Practice-repo/.gi
t/COMMIT_EDITMSG to DOS format...
dos2unix: converting file C:/Users/DivakarC/Desktop/Git-Practice/Practice-repo/.git/COMMIT_EDITMSG to Unix format...
[main 5ad1f1d] Working with stash, relog and cherry-pick
 Date: Thu May 22 14:17:18 2025 +0530
```

3. Blame in git:

   git blame lets you see who last modified each line of a file, when they did it, and what the commit hash was.

   git blame <file-name>

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git blame Textfile2.txt
728f90c6 (divakar-bytewise 2025-05-19 15:49:30 +0530 1) Hellloooo!!!!!!!!!!!!!!!!!!!!
b5adf8cb (divakar-bytewise 2025-05-22 14:17:18 +0530 2) Hiiiiii
b5adf8cb (divakar-bytewise 2025-05-22 14:17:18 +0530 3) Hii diva!!!
b5adf8cb (divakar-bytewise 2025-05-22 14:17:18 +0530 4) Hello man!!!
```

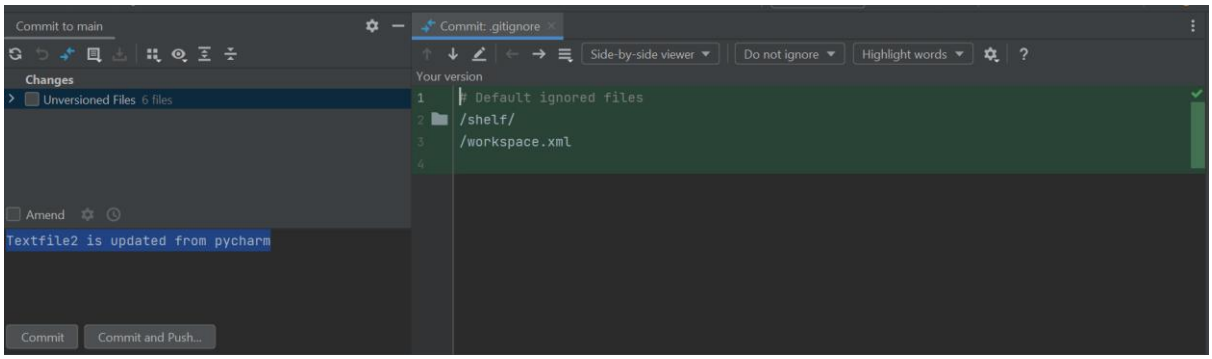
4. Git Features in PyCharm

- PyCharm has excellent Git integration with powerful version control features built-in.

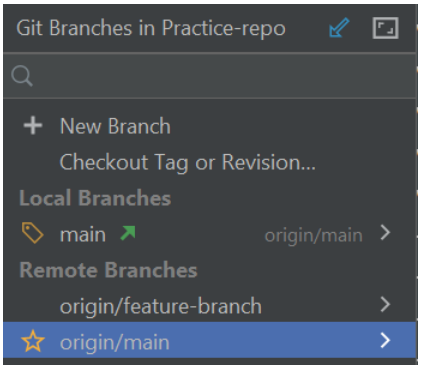
1. **Integrated Development**
   Everything (code editor, debugger, Git, terminal, environment) is in one place.
2. **Powerful Git UI**
   No need for command-line Git — intuitive UI helps with all Git tasks.
3. **Code Intelligence**
   Smart code completion, error detection, and refactoring make coding faster.
4. **Integrated Terminal**
   Run Git commands alongside your project in the same window.
5. **Python-specific Tools**
   Debugger, profiler, test runners, Jupyter support, Django integration, etc.
6. **Virtual Environment Management**
   Easily manage Python interpreters and packages for different projects.
7. **Collaboration Tools**
   Supports GitHub, GitLab, Bitbucket integration for PRs and issue tracking.

Features and Snapshot:

1. Version Control:
   PyCharm shows Git changes visually, lets you commit, push, pull, and manage branches easily.
2. Diff Viewer:
   View side-by-side differences between file versions before committing.

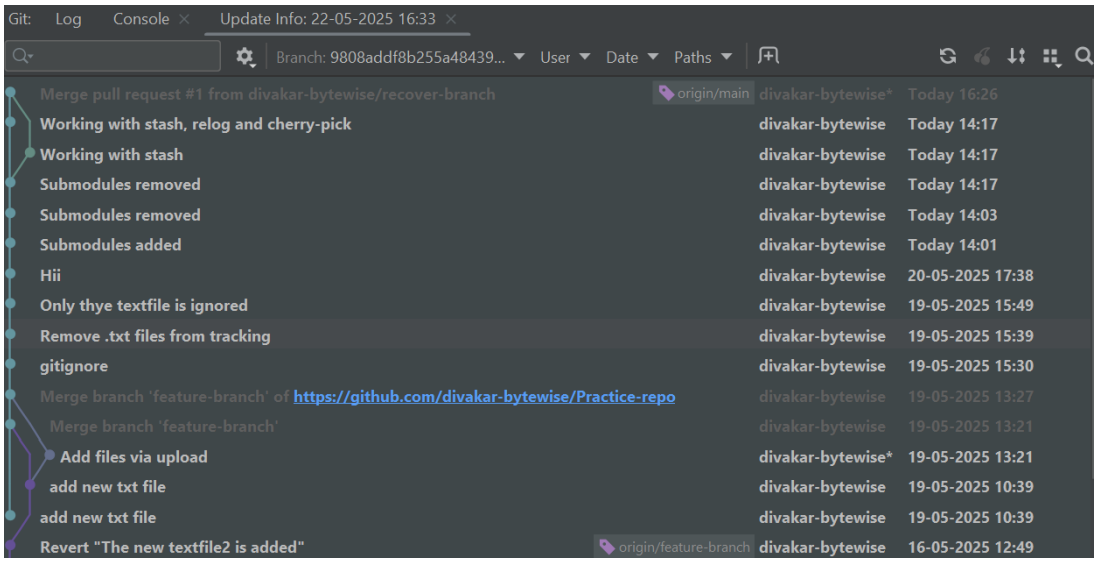


3. Branch Management:
   Easily create, switch, merge, or delete branches.



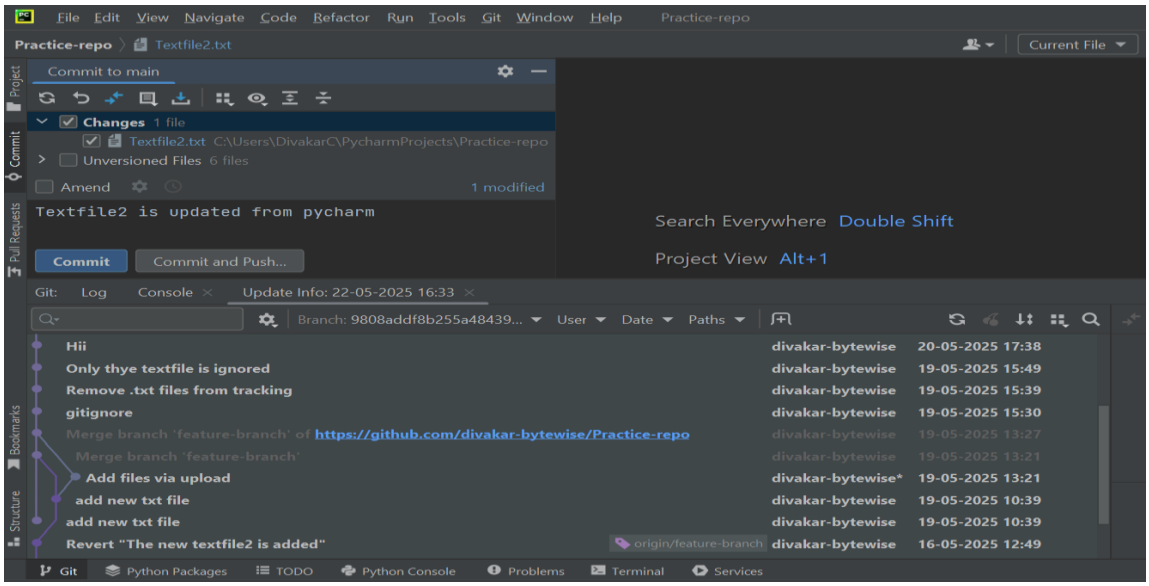
4. Log / History:

   View Git commit history and explore file-specific changes.



5. Commit Tool Window:

   Stage, unstage, and commit changes with a GUI. You can also sign-off and amend commits.

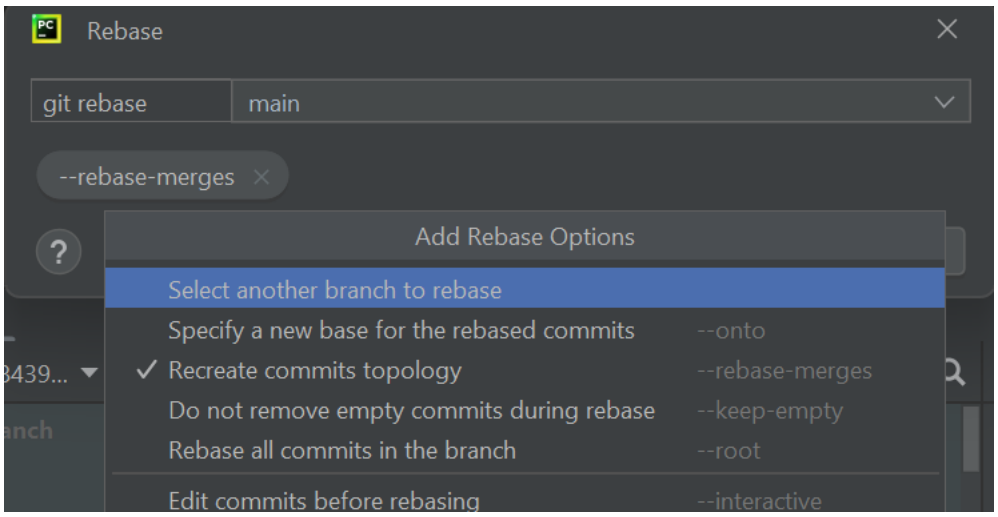


6. Conflict Resolution:

   Graphical interface to resolve merge conflicts with ease.

7. Interactive Rebase:

   Clean up your commit history using rebase from the UI.

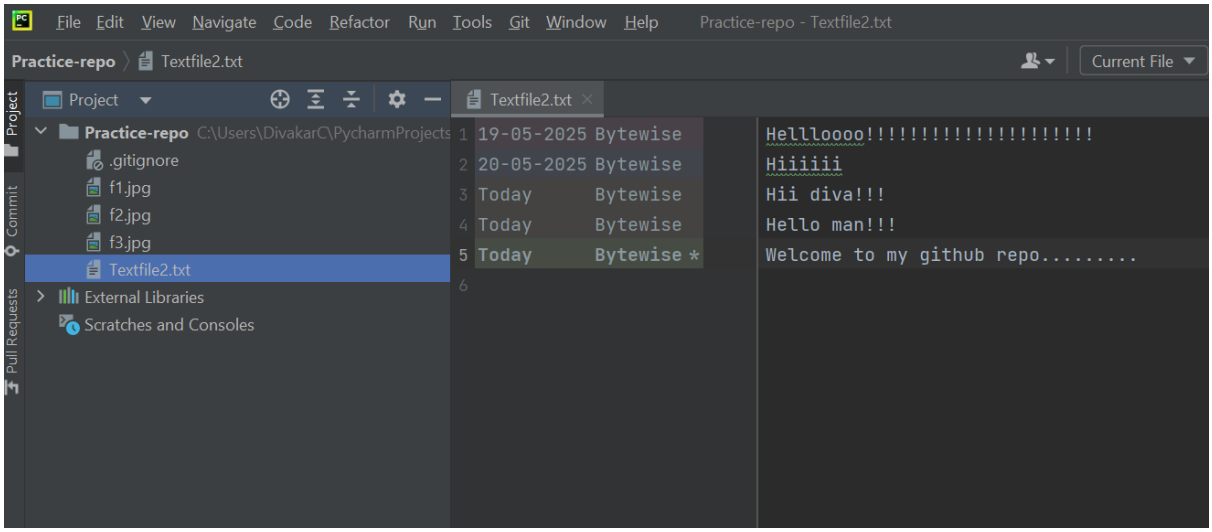


8. Git Stash:

   Save uncommitted changes temporarily and reapply later.

9. Git Blame:

   View who last changed each line of code directly in the editor.

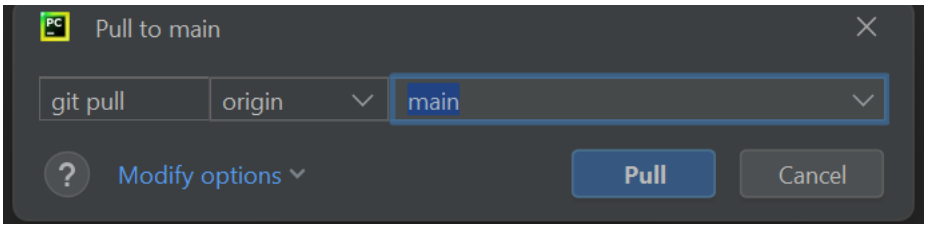


10. Cherry-Pick:

    Apply commits from one branch to another via UI.

11. Pull Requests:

    Direct integration with GitHub and GitLab lets you create, review, and merge PRs.



Date: 23-05-2025

4. Git cherry-pick:

git cherry-pick is a command that applies the changes introduced by an existing commit to the current working branch, regardless of its origin.

git cherry-pick <commit-hash>

```
C:\Users\DivakarC\Desktop\Practice-repository>git cherry-pick 1c55a99
Auto-merging text.txt
[main d880ab4] "working on revert"
 Date: Fri May 23 18:41:38 2025 +0530
 1 file changed, 2 insertions(+), 1 deletion(-)

C:\Users\DivakarC\Desktop\Practice-repository>git log --oneline
d880ab4 (HEAD -> main) "working on revert"
9ee02c1 (origin/main, origin/HEAD) Cheery-pick update
6972b19 "working on revert"
82c8c03 Text file updated
c08a1ba Text file updated
ca3a218 Added a text file
4e252f1 Initial commit
```

5. Git interactive rebase (Practicing):

Interactive Rebase is a Git feature that allows you to edit, reorder, squash, or delete commits in a controlled way, providing a clean and organized commit history.

git rebase -i HEAD~<n>

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git rebase -i 767651d
hint: Waiting for your editor to close the file... unix2dos: converting file C:/Users/DivakarC/Desktop/Git-Practice/Practice-repo/.gi
t/rebase-merge/git-rebase-todo to DOS format...
dos2unix: converting file C:/Users/DivakarC/Desktop/Git-Practice/Practice-repo/.git/rebase-merge/git-rebase-todo to Unix format...
Stopped at 767651d...  first edit
You can amend the commit now, with

  git commit --amend

Once you are satisfied with your changes, run

  git rebase --continue
```

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git log --oneline
d9225b0 (HEAD -> main, origin/main, origin/HEAD) Merge branch 'main' of https://github.com/divakar-bytewise/Practice-repo
b881583 Working with stash, relog and cherry-pick
283049e Working with stash, relog and cherry-pick
acbf444 Combined the commit
a23e666 Working with reword
d50daf1 Git reword in IR
767651d reword b5adf8c re-word
e0b2919 Merge pull request #2 from divakar-bytewise/feature
769fdb0 (origin/feature, feature) Merge branch 'feature' of https://github.com/divakar-bytewise/Practice-repo into feature
57fce04 Hii
59752ff Only thye textfile is ignored
fc924b8 Remove .txt files from tracking
66c2068 Merge branch 'feature' of https://github.com/divakar-bytewise/Practice-repo into feature
75f90e1 working on edit
3f3fba1 git ignore file
ce2cc69 git ignore file
d7f9ae8 Merge branch 'main' of https://github.com/divakar-bytewise/Practice-repo
9fa2f55 Working with stash, relog and cherry-pick
f6283a0 Combined the commit
5ad1f1d Working with stash, relog and cherry-pick
b5adf8c Submodules removed
44d6fe3 Submodules removed
f645f6a Submodules added
658c819 Hii
728f90c (tag: v1.0, origin/feature-branch, feature-branch) Only thye textfile is ignored
56e4be9 Remove .txt files from tracking
fa929dc gitignore
03a0242 (upstream/main) Merge branch 'feature-branch' of https://github.com/divakar-bytewise/Practice-repo
7f9ac12 Merge branch 'feature-branch'
7a4ac5b (upstream/feature-branch) Add files via upload
3b4d3e1 add new txt file
5279fe2 add new txt file
```

https://github.com/divakar-bytewise/Practice-repo
Ctrl+Click to follow link

In the opened editor:

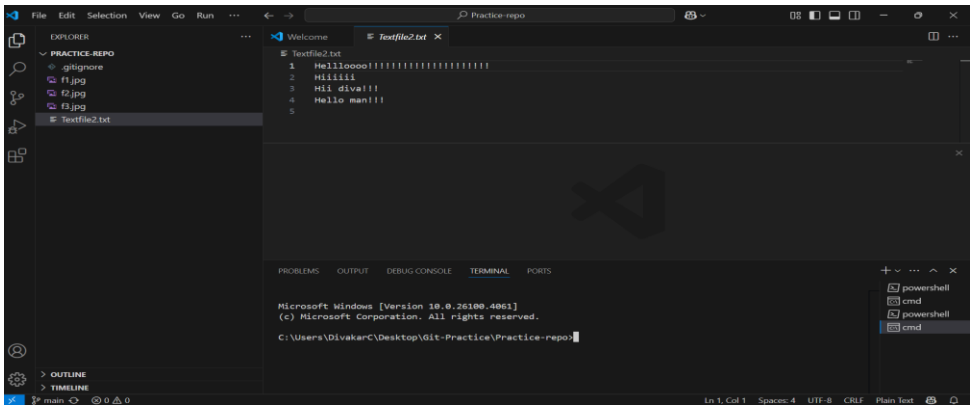
- Use pick to keep as-is.
- Use reword to change commit message.
- Use squash to merge commits.
- Use drop to delete a commit.

6. Vs Code:

Using **Git in Visual Studio Code (VS Code)** is simple and powerful.

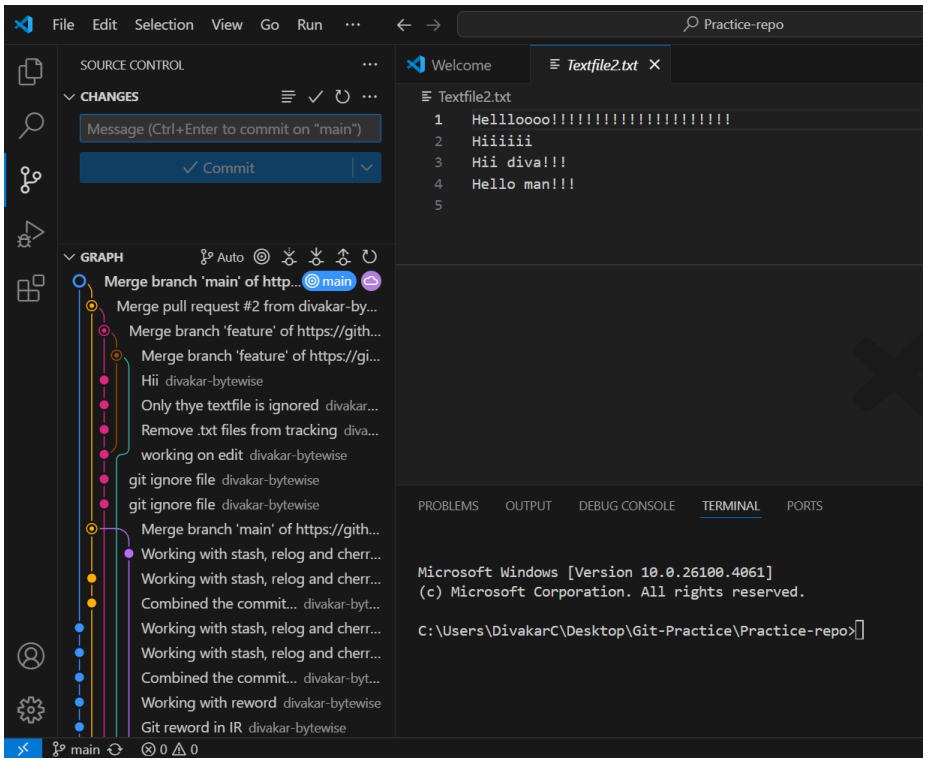
1. Opening a Project Folder

Click on File → Open Folder and select your Git project.



1. Git Features in VS Code (Source Control Panel)

   Click the Source Control icon ( ) in the Activity Bar or press Ctrl+Shift+G.



2. Git Terminal Commands in VS Code:

   git status

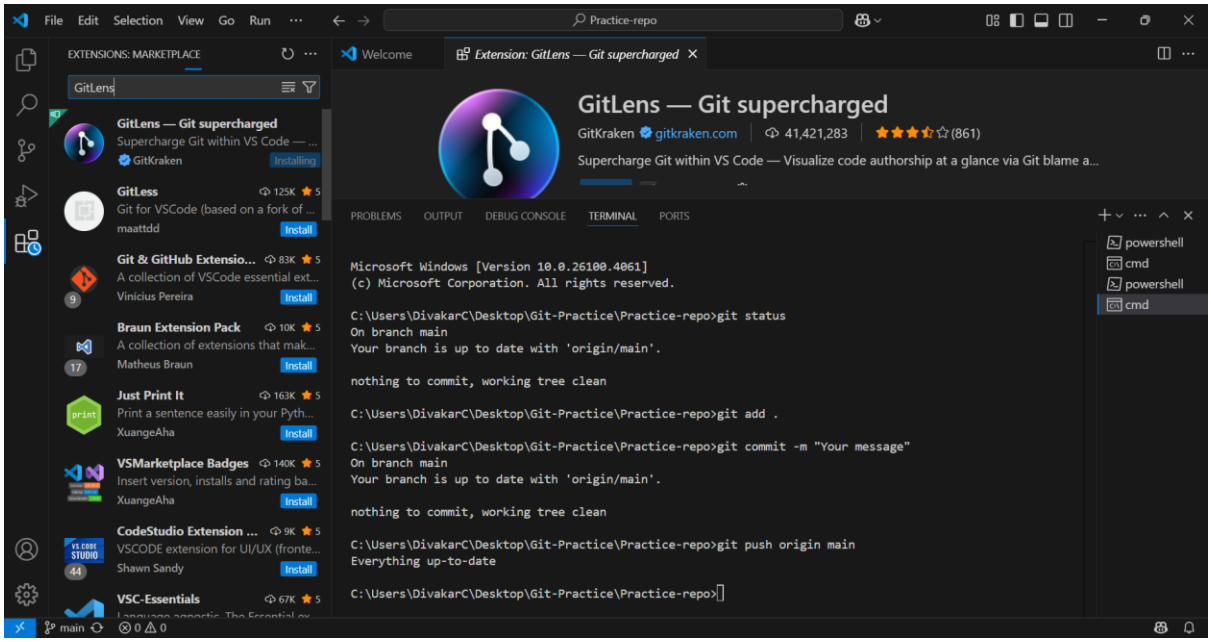
   git add .

   git commit -m "Your message"

   git push origin main

```
C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git add .

C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git commit -m "Your message"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

C:\Users\DivakarC\Desktop\Git-Practice\Practice-repo>git push origin main
Everything up-to-date
```

3. GitLens Extension:
    - Blame annotations
    - Commit history
    - Branch comparison
    - Line-by-line author tracking



Summary:

VS Code is a full-featured development environment best for writing, managing, and versioning code. It simplifies complex Git tasks through GUI. Command Prompt, while powerful for quick tasks and scripts, is less user-friendly for code development and version control workflows.