LECTURE -1

Contents

- What is DataOps?
- Roles of Data Engineer and Operation Team
- Tools going to Learn in DataOps

Data Engineer

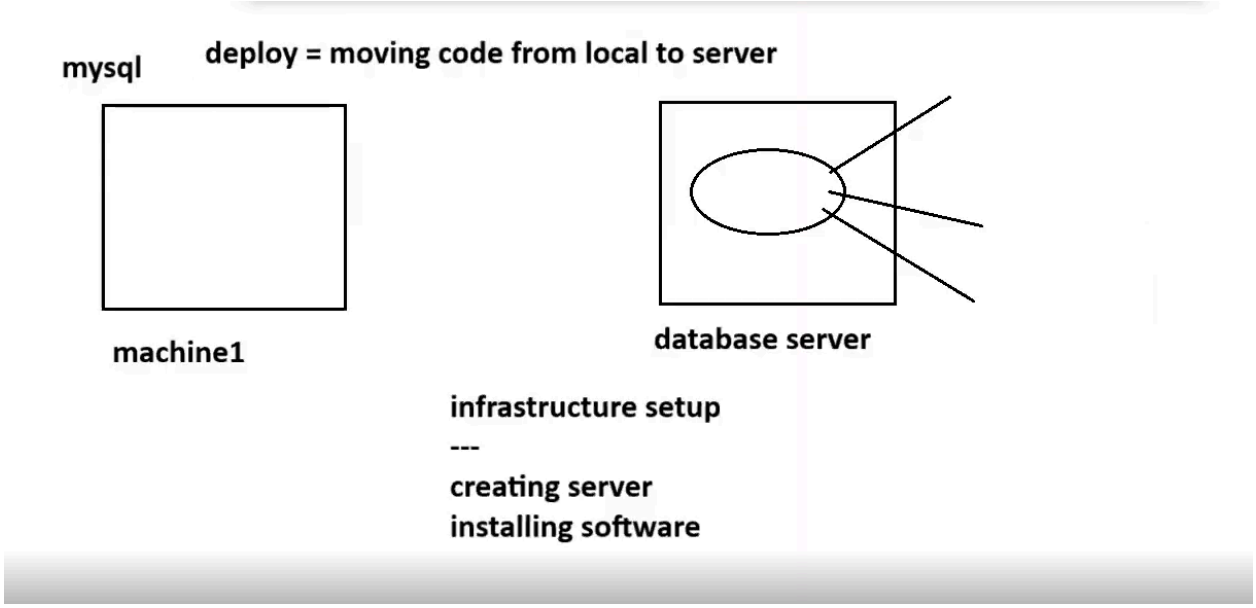
- Develop ETL Pipeline/ Automation Script
- Collaborate with other Developers in Team
- Maintain Source Code Repos

Operations Team

- Provide the required Infrastructure
- Deploy the Application and Database.
- Monitor application

Tools

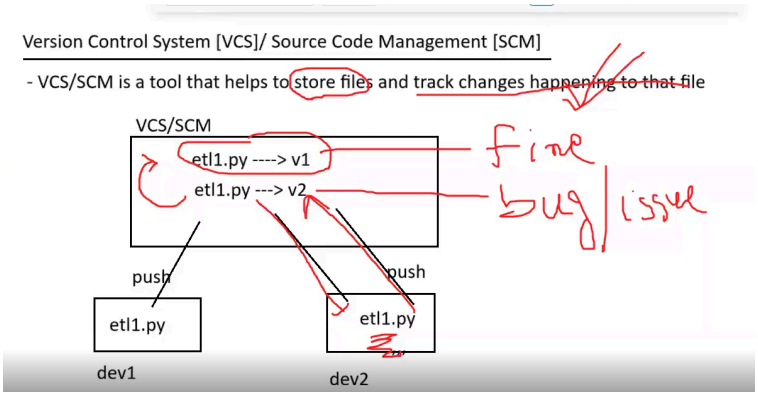
1. GitHub
    - Source code management tool
2. Bitbucket
    - Source code management tool
3. Confluence
    - documentation tool
4. Jira
    - Project management tool
5. GitHub Action
    - CI CD tools
6. Jenkins
    - CI CD tools



LECTURE = 2

Version Control System

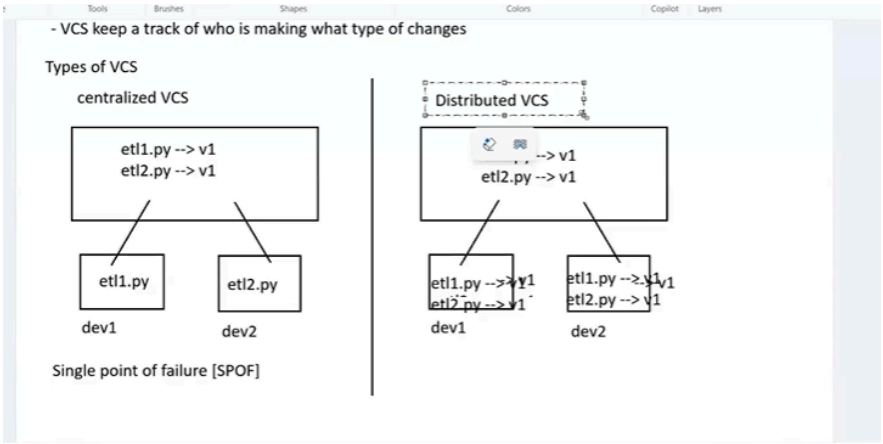
- Version Control System (VCS) / Source Code Management (SCM) is a tool that helps to store files and track the changes happening to files.
  - dev1 push the code in vcs for
  -vcs will store code as well as version of dev1
  -also dev2 version store
  -why it store multiple versions = coz when it have any bug ,issue, it can change by other dev2 separately
  - store n number of versions



- VCS also preserves older and later versions of the code so that at any time we can switch between whichever version we want.
- VCS also keep a track of who is making what kind of changes
- Types of VCS
    - Centralized VCS
    - Distributed VCS

Centralized VCS

- In centralized VCS, a central repo is maintained where all the versioned files are kept.
- Developers can check-in and check-out changes from their own computers.
- Example:
    - SVN

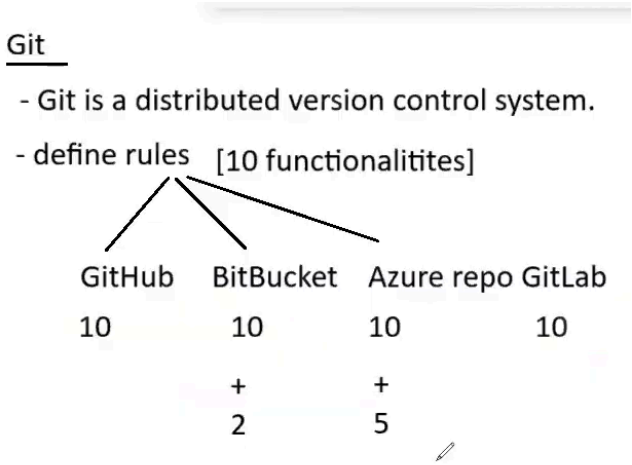


Disadvantages of Centralized VCS

- In case of central server failure the whole system goes down.

Distributed VCS

- We have a local repository on every team member's machines where version controlling happens at the level of individual team members from where it is uploaded into a remote server where version controlling happens for the entire team.
- Developers will perform PUSH and PULL operations.
- Example:
    - Git
    - Git is distributed version control system
    - Git will track who is made what kind of changes

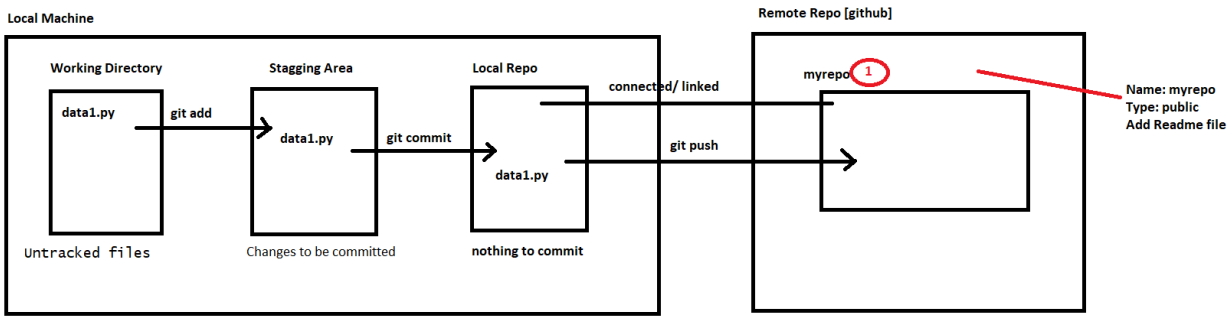


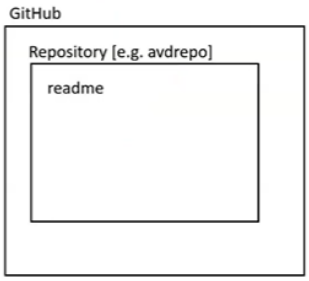
-

Setting up git on Windows

1. Download git from https://git-scm.com/downloads
2. Install it
3. Open Git Bash and Execute the git commands
    - Git –version
    - Git config –global user.name “kunal savade”

Git Life Cycle Phases / Workflow





visibility

--

public

anyone can download code

they will not able to commit [acccess]

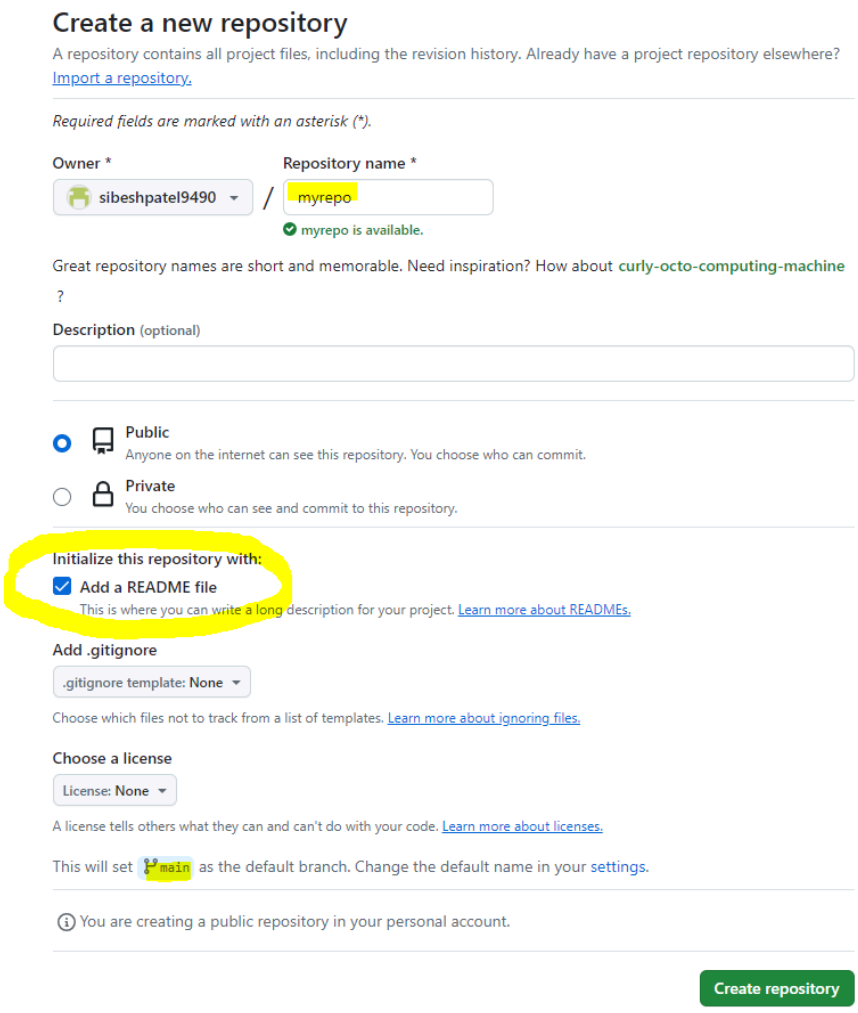
private

download
push code

[access]

Git visibility is private any one can download and commit changes and then push the code

Create Remote Repo



- Repository Name: my-repo
- Type: Public
- Add Read me file [enabled]

Local Machine

Configuring User Name and Email

git config --global user.name "Sibesh Patel"

git config --global user.email "sibeshpatel9490@gmail.com"

Create Repo

Step1:

- Initialize git repository

git init

- Pull code from remote repo
  -mgit pull git-repo-url

Step2:

- Create few files
  data1.txt
- Check Status
  git status

Step3:

- Move files from working directory to staging area
  git add .

Step4:

- Move files from staging area to local repo
  git commit -m 'data1.txt added'

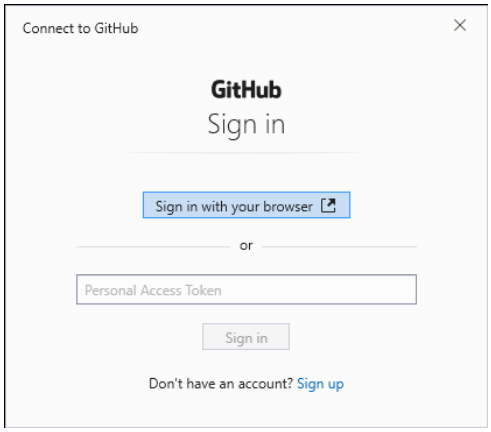
Step5:

- Link with remote repo
  git remote add origin <git-repo-url>
  git branch -M main

Step6:git

- Push Code to Remote Repo first time

  git push -u origin main
- It will ask for authentication.



Click on sign-in with browser and login to GitHub for authentication.

LECTURE = 3

git clone

```
[core]
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
[remote "origin"]
        url = https://github.com/kunalsavade-sys/akash11.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = origin
        merge = refs/heads/main
```

- This will download all the code from the remote repository into the local repository. (like gitpull)
- Automatically create .git folder
- Establish link to remote repo (github)
- It update link to remote repo
- It updated branch as main
- It is generally used only once when all the team members want a copy of the same code.
- Syntax:

git clone remote_git_repo_url

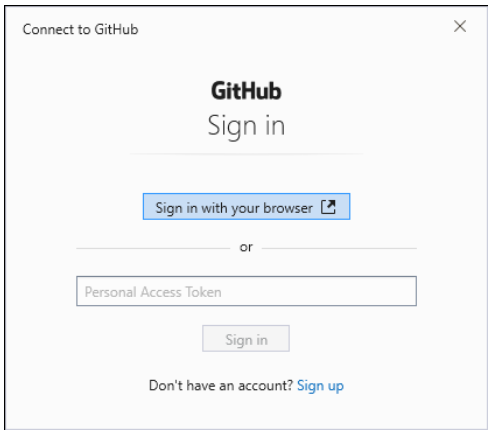
git pull

- Syntax:
  git pull <remote-repo-url>
- Get all latest data

Touch file = untract file and location is working stage

Different ways of Authentication

- It will ask for authentication.



- Create Personal Access Token
  My Profile Settings -> Developer Setting Personal Access Token (classic)

Different ways to add file into staging area

git add filename1
git add filename1 filename2
git add . [all files]
git add *.txt [all text file]

Working with folders during commit

mkdir demo
Touch demo/a.txt demo/b.txt
git add . = move all file

git commit –m “demo added”
git push

Rename sathi

- Change first into local file system
- Git status made = you will get deleted file and renamed file as untracked
- Git add.
- Git commit -m “renamed app1 ⟶app11”
- Git push
- You will change name in remote repository

git ignore

- To ignore some files while committing
- Steps
  - Touch gitignore
  - Check in LFS
  - THERE you can write on note pad
  - If i write a.logs == then it will ignore that file in git status
  - *.logs = all log file ignore
  - folder/ = to ignore folder
  - Now log and folder not in working area
  - If main.log i want in working area
  - !main.log

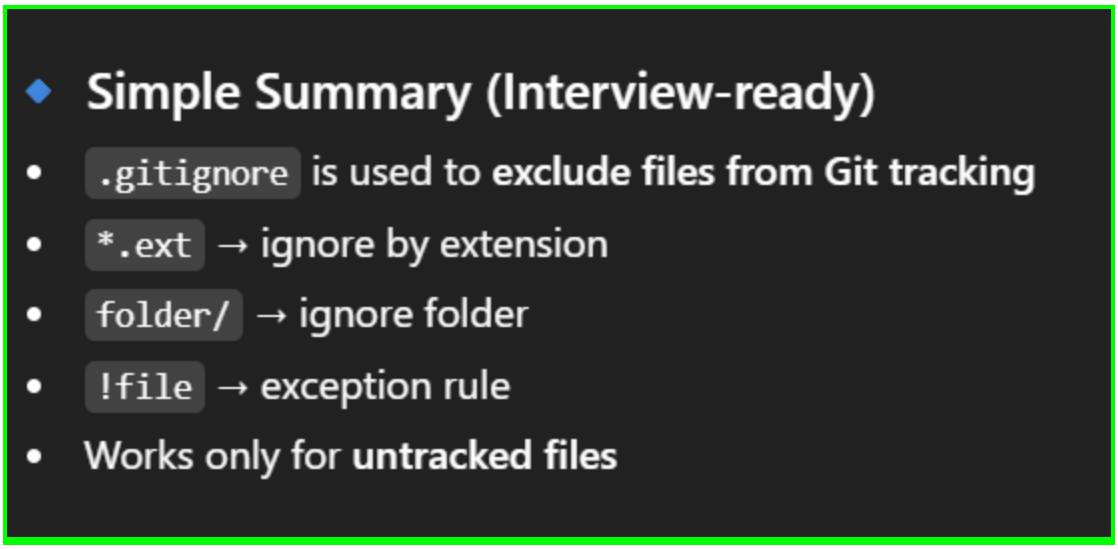
touch .gitignore (creates a .gitignore file)

logs/ #completely ignore the folder & content inside

*.class #completely ignores files ending with .class

*.txt # completely ignores files ending with .txt

!test1.class # don’t ignore test1.class

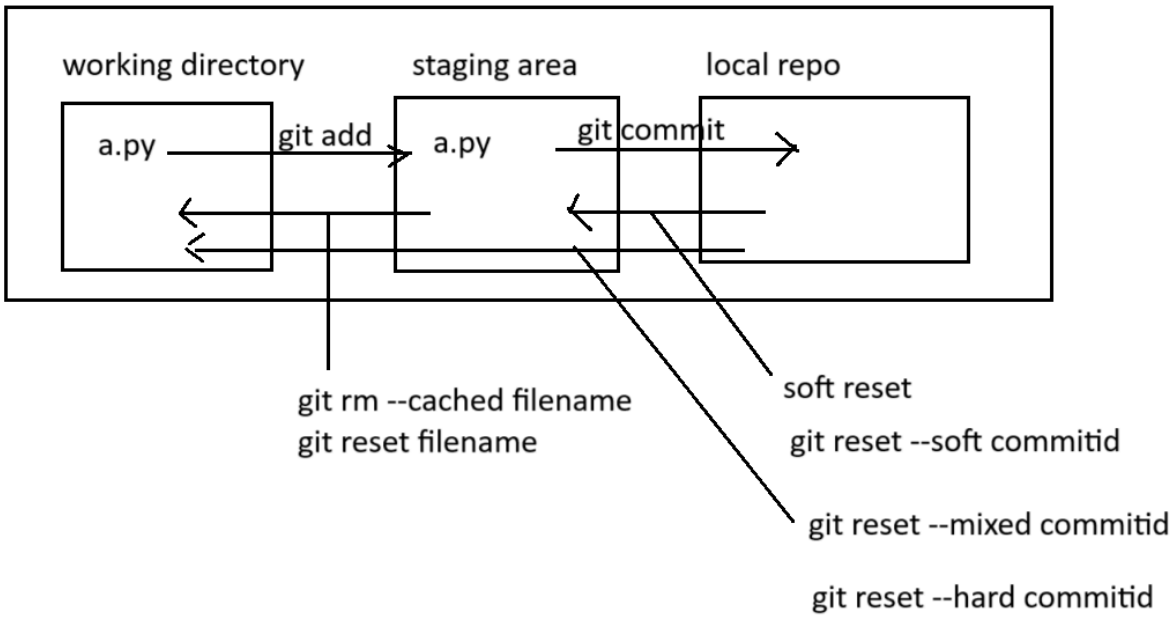


Lecture -4

Contents

- Git reset
- Git tag
- Working with branch

Git Reset



Git reset - soft commitid - we can not use directly due to

- **We need before this commit**
- **Local repo madhun staging area madhe move karnyasathi = previous commit id**
- **We get our file in staging area**
-

I want file from local to working

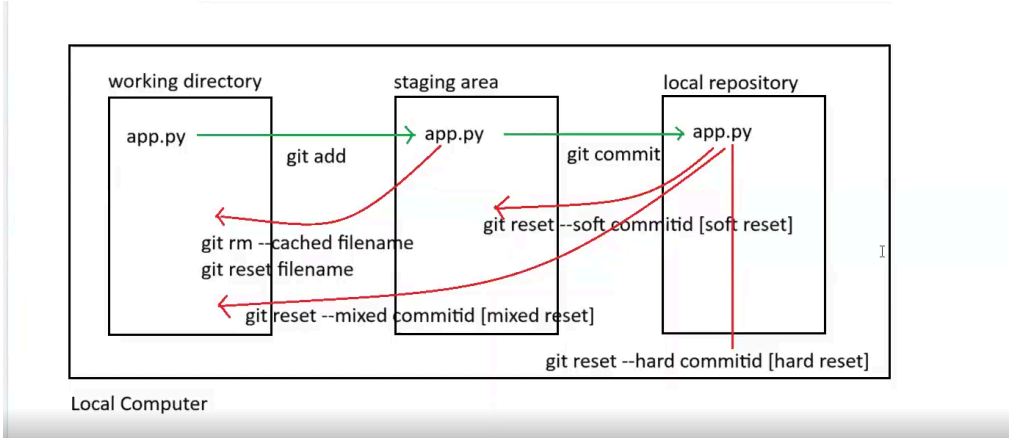
- **Mixed reset**
- **Git reset - mixed previous commit id**

First move your readme file from remote repo to local repo then you can add anything to staging area

I want to delete file log detail from local file system and also from local repository

- **Git reset - hard commitid**

Git Reset



- It is used toggle between multiple versions of git and access whichever version we want.
- Reset can be done in 3 ways
    - Hard reset
    - Soft reset
    - Mixed reset

Hard Reset

- In hard reset HEAD simply points to an older commit and we can see the data as present at the time of that older commit
- It will delete data from working directory.
- Syntax:
  git reset --hard commit_id

Soft Reset

- Soft reset will move the head to an older commit.
- Files will be present in the staging area.
- Syntax:
  git reset --soft commitid

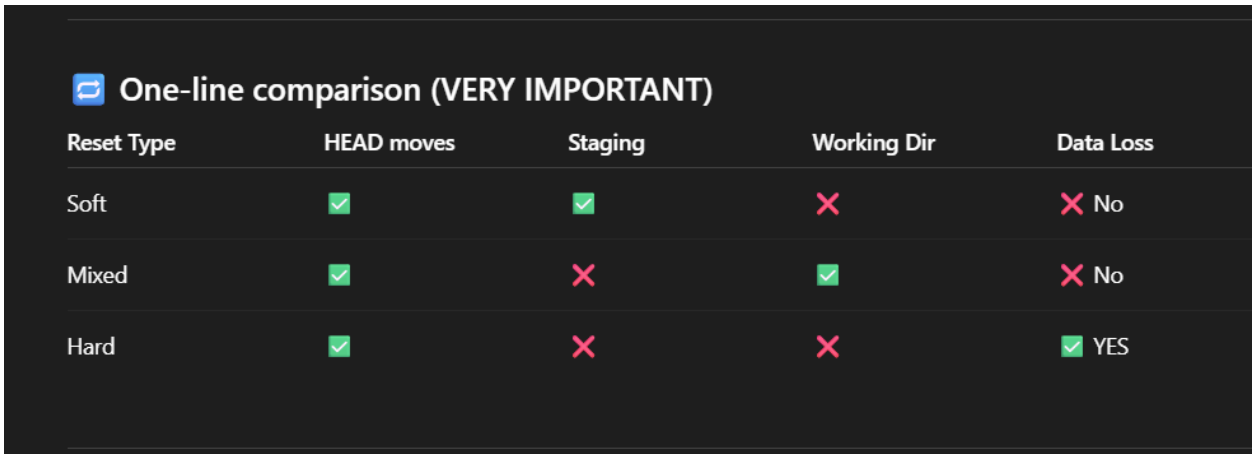
Mixed Reset

- Mixed reset moves the head to an older commit.
- The files will be present in the untracked/modified section
  git reset --mixed commitid

Work with git tags

- Git tags are aliases for commit ids.

- For convenience
- User friendly name
- Tags are always created against specific commit ids
- lists all available tags



🔁 One-line comparison (VERY IMPORTANT)

Reset Type	HEAD moves	Staging	Working Dir	Data Loss
Soft	✅	✅	❌	❌ No
Mixed	✅	❌	✅	❌ No
Hard	✅	❌	❌	✅ YES

- **For latest commit id we will use**
    - Git tag -a v1.0 -m“v1.0 tag added”{latest commit}
    - Git tag -a v1.0 -m“v1.0 tag added”{latest commit} = **for specific commit**
    -

git tag

- git show v1.2
    - shows which commit id tagged, who committed & what is committed
- git tag --a v1.2 -m "any message"
    - by default git tags the recent commit id / last commit id
- git tag --a v1.3 <commit id> -m "any message"
    - tags a specific commit id provided

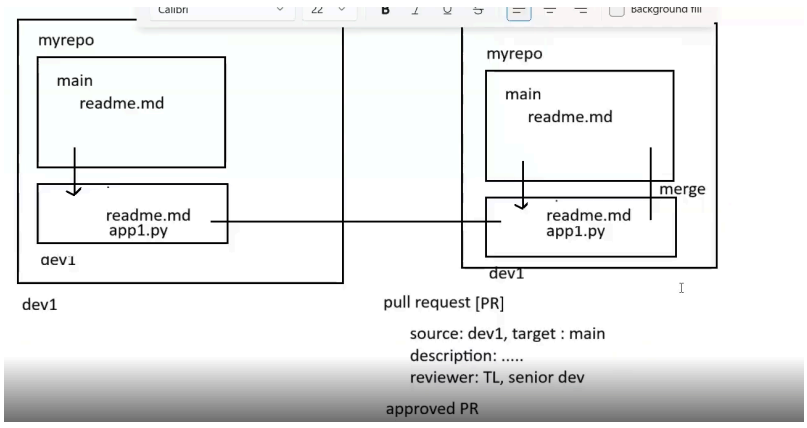
Branching in Git

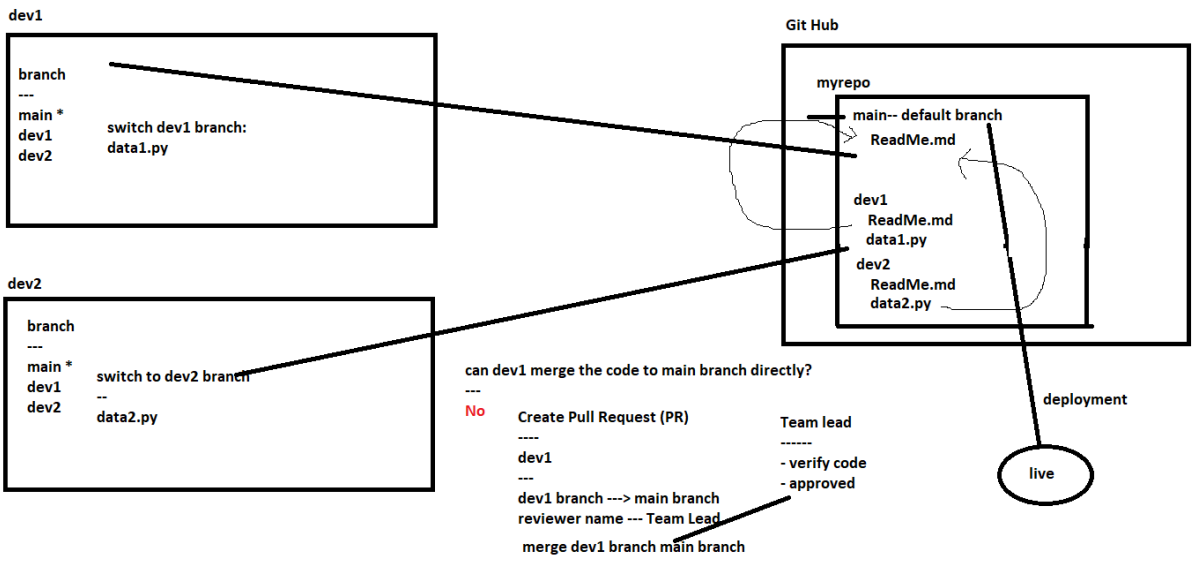
Main branch to child branch dev1 madhe

we clone the code then dev2 = he will make file then = again shift it to dev1 child branch

Then merger it

Before that = pull request (PR)



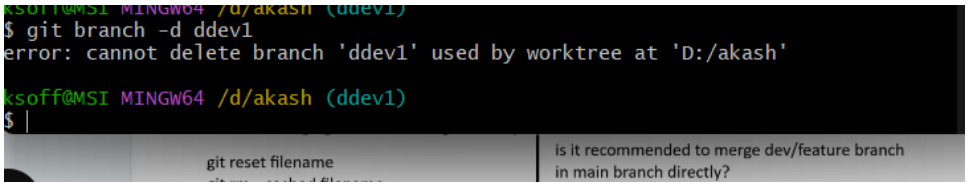


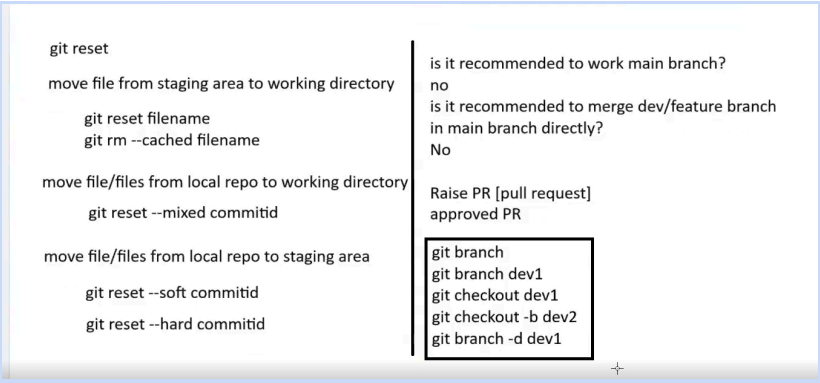
- Branch is a copy of main branch at certain stage which will be used to develop certain features.
- We can create separate branches for different functionalities and later merge them with the main branch.
- To see the list of local branches

```
HP@LAPTOP-4D9KAJCM MINGW64 /e/Programs/Git Examples/avdrepo5 (main)
$ git branch
* main

HP@LAPTOP-4D9KAJCM MINGW64 /e/Programs/Git Examples/avdrepo5 (main)
$ ls
README.md  a.py  b.txt

HP@LAPTOP-4D9KAJCM MINGW64 /e/Programs/Git Examples/avdrepo5 (main)
$ git branch dev1

HP@LAPTOP-4D9KAJCM MINGW64 /e/Programs/Git Examples/avdrepo5 (main)
$ git checkout dev1
Switched to branch 'dev1'

HP@LAPTOP-4D9KAJCM MINGW64 /e/Programs/Git Examples/avdrepo5 (dev1)
$ ls
README.md  a.py  b.txt
```

1. git branch dev1= to create a branch = along with main branch
2. Git branch = list all branches

3. Git checkout dev1 = IT NEED TO EXIST BRANCH = switched to branch dev1 = every file from main to dev1
4. To cheakout main branch file = git cheakout main
5. New file in dev1 not in main
6. Git checkout -b dev2 = create and switch it automatically
7. Git branch -d dev 2 = = we can not delete it directly == karan not merged
8. Git branch -d dev1 = delete





LECTURE - 5

Content

When we create a child branch in it , if we have file .

- when we use checkout command file get copied in main branch

(file is in working stage)

- When we commit a file it goes to the local repository .

- Now if we use the checkout command file will not get into the main branch.

(file is in local repository)

- If it is in the working stage then you can watch it everywhere .
- But once we commit ii then we will not get it .
- First fresh child branch get all files of main branch
- But after that it will copied due to commit

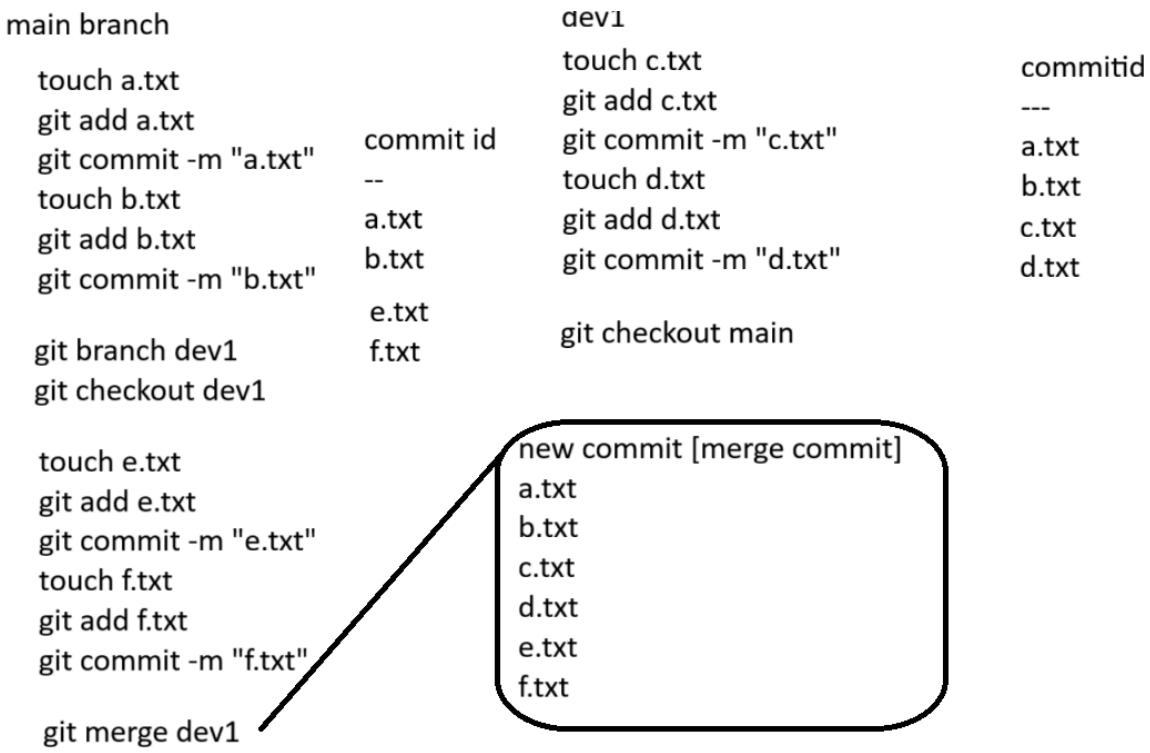
Content

3 merging technique

- Git Merge
- Git Merge with Squash
- Git Merge with Rebase

Git Merge

- Merge keeps all commit history and creates merge commits.

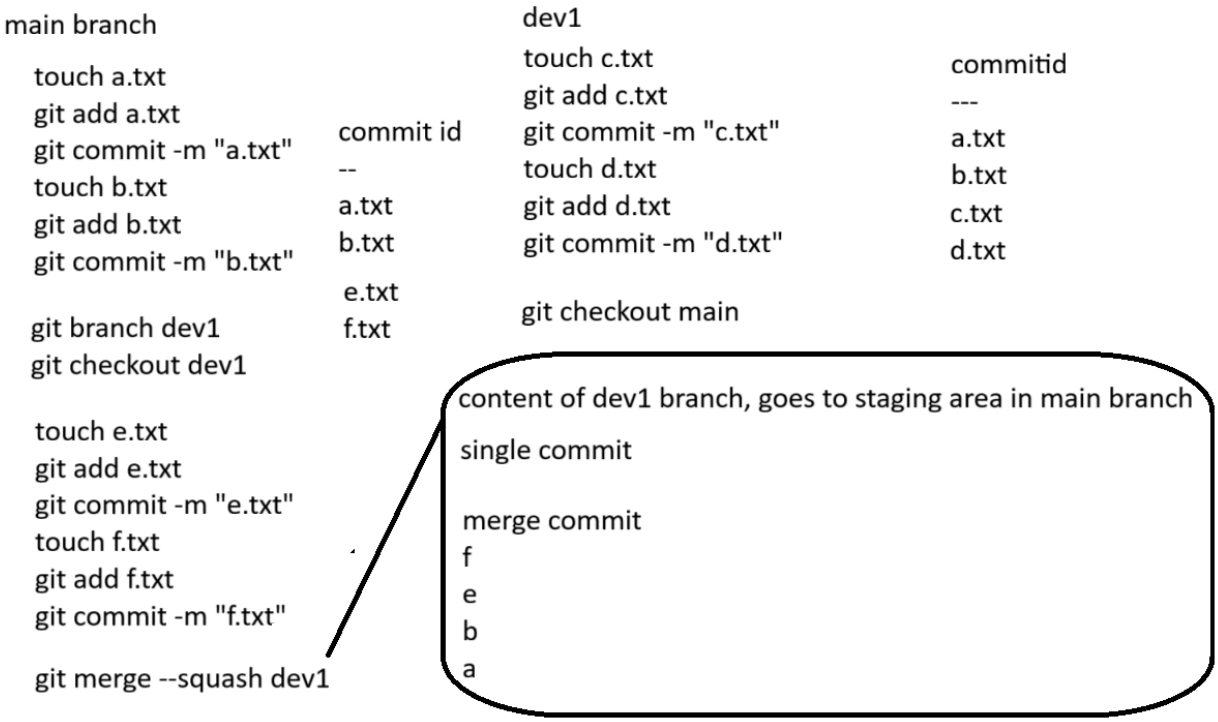


1. In Hard delete = main origin file will stay and other will get delete
2. After merging = 1 more commit id will be added t

Git merge dev1 = from main branch

Git Merge with Squash

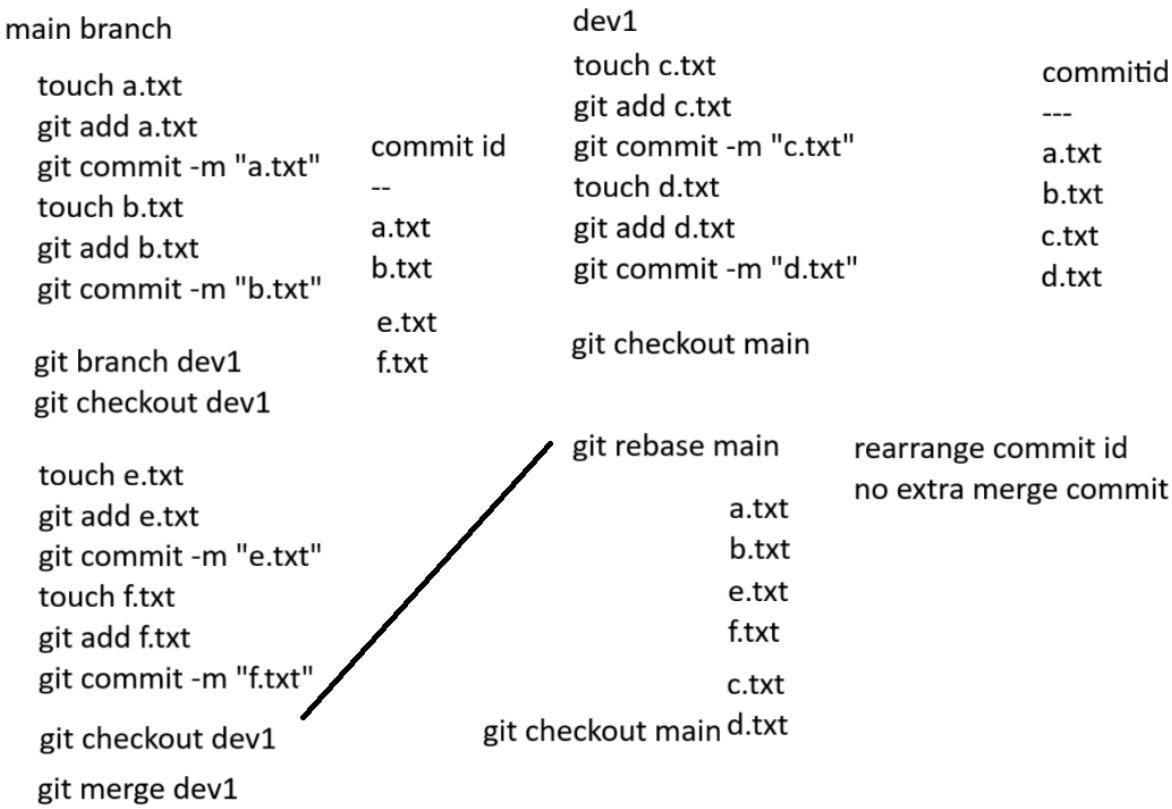
- Squash Merge simplifies commit history into a single commit.



- git merge –squash branch1
  Git has staged the changes from the branch we are merging (branch1), but it hasn't created a commit yet.
- git status
- git commit -m "commit message"
- Reduced commit id
- Move to staging area not in local repository
- Checking files sathi == 2 commadd == GIT status & ls (to check files in local repository)

Git rebase

- This is called as fast forward merge where the commits coming from a branch are projected as the top most commits on master branch git
-



-

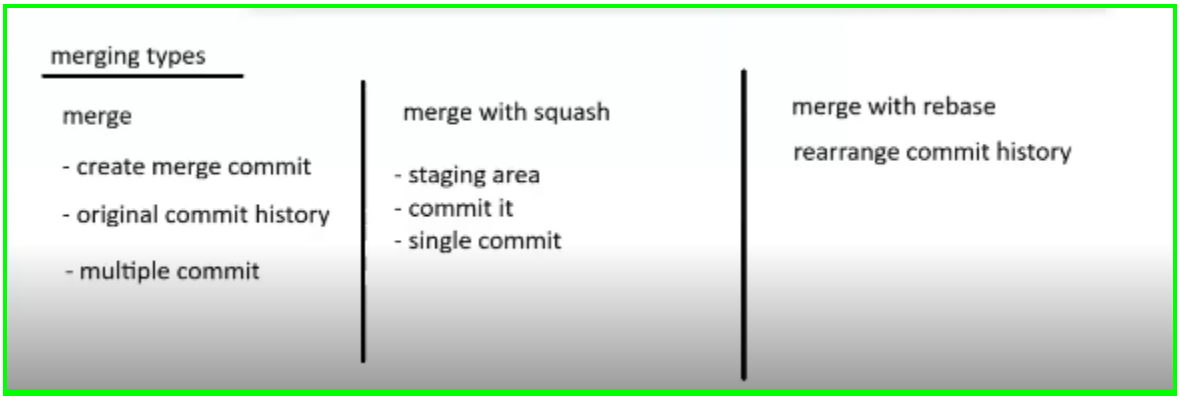
- git checkout branch1
- git rebase main
- git checkout main
- git merge branch1

```
$ git log --oneline
3f75f1c (HEAD -> dev1) d.txt
a34a7ed c.txt
8b7defa b.txt
9a887b2 a.txt
91296ce (origin/main, origin/HEAD) Initial commit

HP@LAPTOP-4D9KAJCM MINGW64 /e/Programs/Git Examples/avdrepo1 (dev1)
$ git rebase main
Successfully rebased and updated refs/heads/dev1.

HP@LAPTOP-4D9KAJCM MINGW64 /e/Programs/Git Examples/avdrepo1 (dev1)
$ git log --oneline
99d7387 (HEAD -> dev1) d.txt
143eb5a c.txt
0276504 (main) f.txt
d1809c2 e.txt
8b7defa b.txt
9a887b2 a.txt
91296ce (origin/main, origin/HEAD) Initial commit
```

- Check commit id of c.txt = it is different before rebase and after rebase
- Again go to main branch and then add merge command == same commit id as dev1 branch

Conclusion

1. After rebase command in child branch you will get NEW REARRANGED COMMIT ID
2. No extra merge id



LECTURE = 6

Merge Conflict

- Occurs when the same file is modified at both branches.

merge conflict

main branch

touch data1.txt
git add data1.txt
git commit -m "data1.txt added"

git branch dev1
git branch dev2

git merge dev1

data1.txt
--
welcome1

git merge dev2

merge conflict

dev1
- edit data1.txt
welcome1
git add data1.txt
git commit -m "data1.txt changed by dev1"

dev2

- edit data1.txt
welcome2
git add data1.txt
git commit -m "data1.txt changed by dev1"

- Create Branch
  git branch branch1
- Switch to branch1
- git checkout branch1

- Create Files
  touch data.txt
- opens the file for editing, enter some text & save the file
- Move Staging Area
  git add .
- Move to Local Repo
  git commit -m "data.txt added"

  git checkout main

  touch data.txt

  opens for editing, enter some text & save it

  git add .

  git commit -m "data.txt added at master"

  git merge defect #Merge Conflict

  Auto-merging data.txt

  CONFLICT (add/add): Merge conflict in data.txt

  Automatic merge failed; fix conflicts and then commit the result.

  git status

  On branch master

  You have unmerged paths.

  (fix conflicts and run "git commit")

  Unmerged paths:

  (use "git add <file>..." to mark resolution)

  git add .

  git commit -m "merge conflict resolved"

  git status

Lecture = 7

Collaborators

- Team Lead/ Project Lead/ DevOps team sends invitations to team members from the central repository.

Add Collaborators

- Go to repository page
- Click on "Settings" > "Manage access".
- Click "Invite a collaborator" and enter their GitHub username/ email.
- Send the invitation.
- Team members will accept the invitation.

Clone Code

- Import repository using git clone command.

Create a Pull Request

- Go to your repository on GitHub.
- Click on the "Pull requests" tab.
- Click the "New pull request" button.
- Select the base branch (e.g., main) and the compare branch (e.g., new-feature-branch).
- Click "Create pull request".
- Fill in the details and submit the pull request.

Review the Code

- Team members review the pull request by adding comments and approving the changes.
- You can address any feedback by making additional commits to the new-feature-branch.

Merge the Pull Request

- Once approved, click the "Merge pull request" button.
- Confirm the merge.
- Delete the branch if desired.

Git Stash

- Git stash command keeps the working copy clean, by storing the partially completed task code in temporary storage and allows bringing them back to working copy whenever we want.

Implementations

- Do some changes in data.py [dev1]
- Git add .
- Got some high priority task, keep ongoing task code in stash mode
  git stash
- Show all stash list
  git stash list
- Complete new task
  - Add to staging area
  - Add to local repo
  - Push to remote repo
- Bring old task back by applying the stash
  git stash apply
- Complete old task
  - Add to staging area
  - Add to local repo
  - Push to remote repo

LECTURE - 8

GUI Tools to Work with Git

- GitHub Desktop
- Source Tree

Amending

- Amending a commit allows you to modify the most recent commit.

- This can be useful if you forgot to include some changes, need to correct a mistake, or want to update the commit message.

Git Revert

- The git revert command is used to create a new commit that undoes the changes made by a previous commit.
- This is a safe way to undo changes because it preserves the history and does not modify the existing commit history.

Cherry-picking

- Cherry-picking in Git allows you to apply the changes introduced by an existing commit from one branch onto another branch.
- This is useful when you want to take a specific commit (or commits) and apply it to another branch without merging the entire branch.

Fork

- Forking creates a personal copy of someone else's repository on your account.