GIT

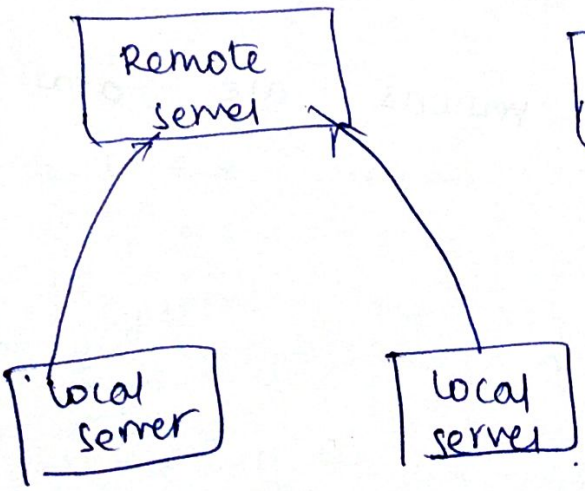
Version control

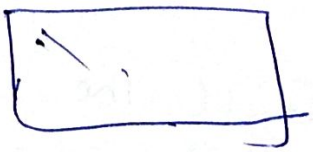
It takes the code coming from all the developers and integrating in to ones.

2) It is preserving older versions of the code & the newer versions of the code so that at any point of time you can switch between the versions

3) It will track who will make changes to the code

centralized version controlling

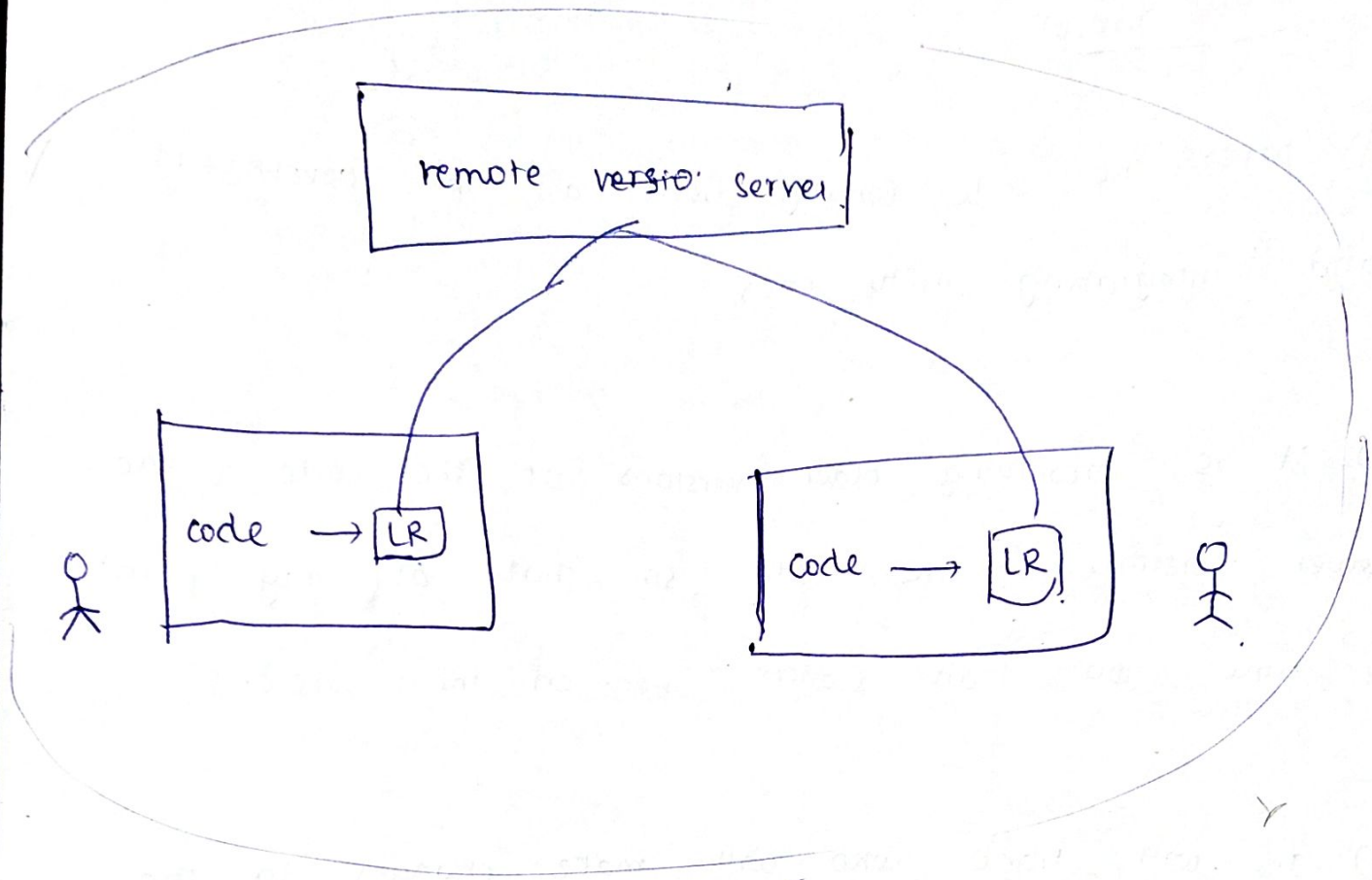




developers sim
upload the cod
in the remote se

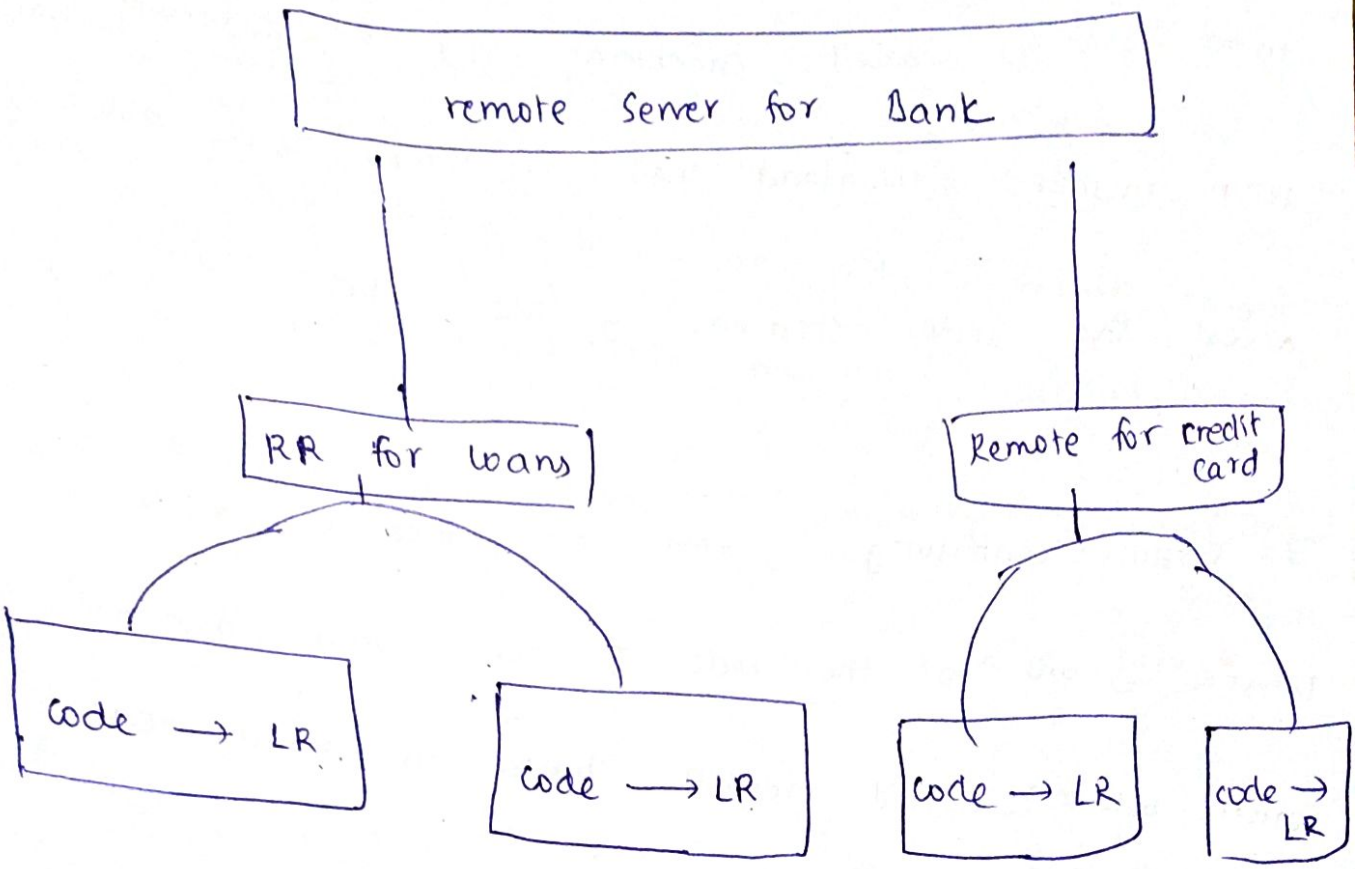
preserving older version
& newer versions.

distributed version controlling



Version controlling at the level of individual team members.

What ever the versions are present in remote server the copies of the versions are available in local repositories.



Version controlling :-

This is the process of maintaining all version of the code.

All the developers upload the code into version controlling systems. the vcs accepts the code uploads and integrates in to one single project. The process of

in to VC's is called Checking. Next time when the team members download the code. they will able to access the code created by the entire team

3) version controlling systems also preserve older & latest versions of the code so that team members can switch between any version based on requirement.

VC's also keep a track of who is making what kind of changes.

VC's are of 2 types

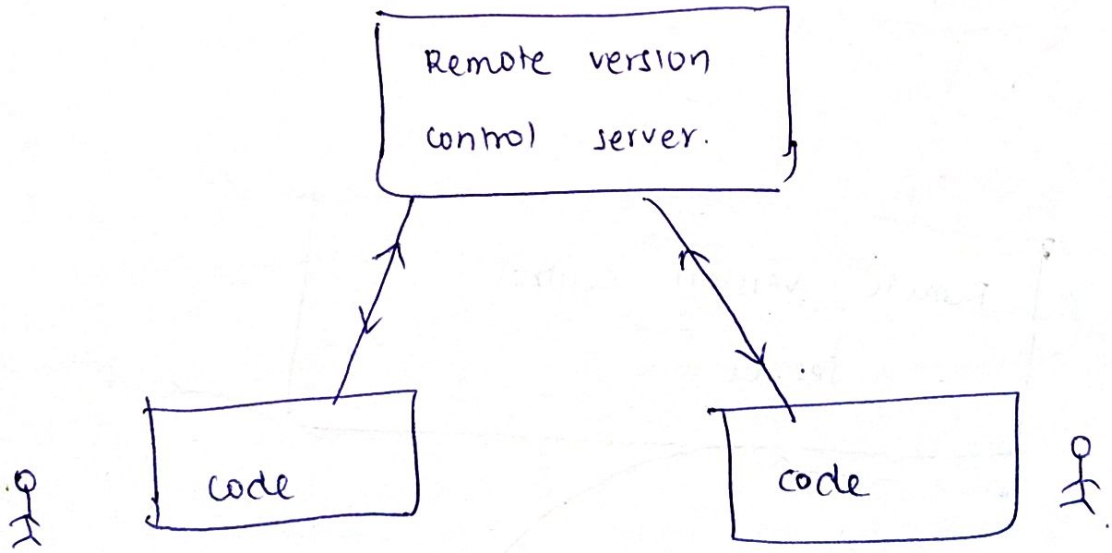
1) centralized version controlling

2) Distributed version controlling.

Centralized version controlling

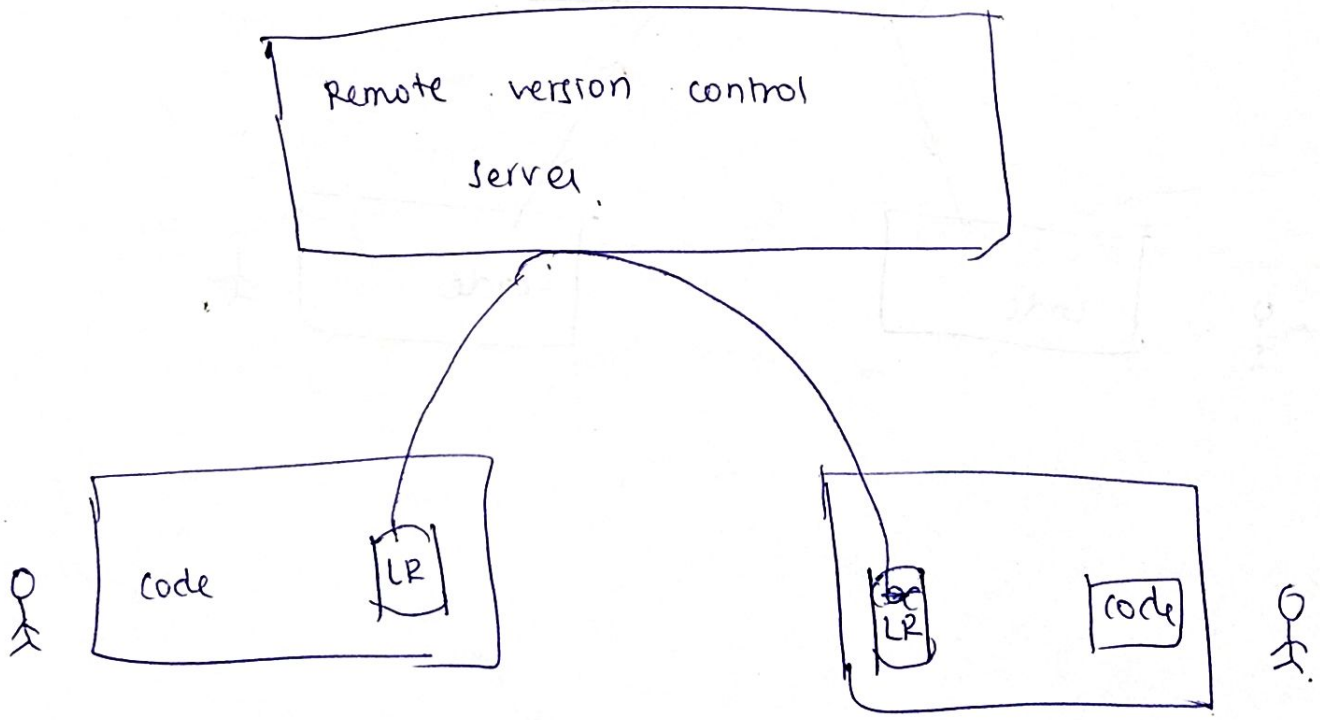
In cvc we have a remote server where version controlling happens. On the individual team members machines only code is present. All the version controlling is performed only once the code is pushed in to remote server.

eg: SVN (sub version)



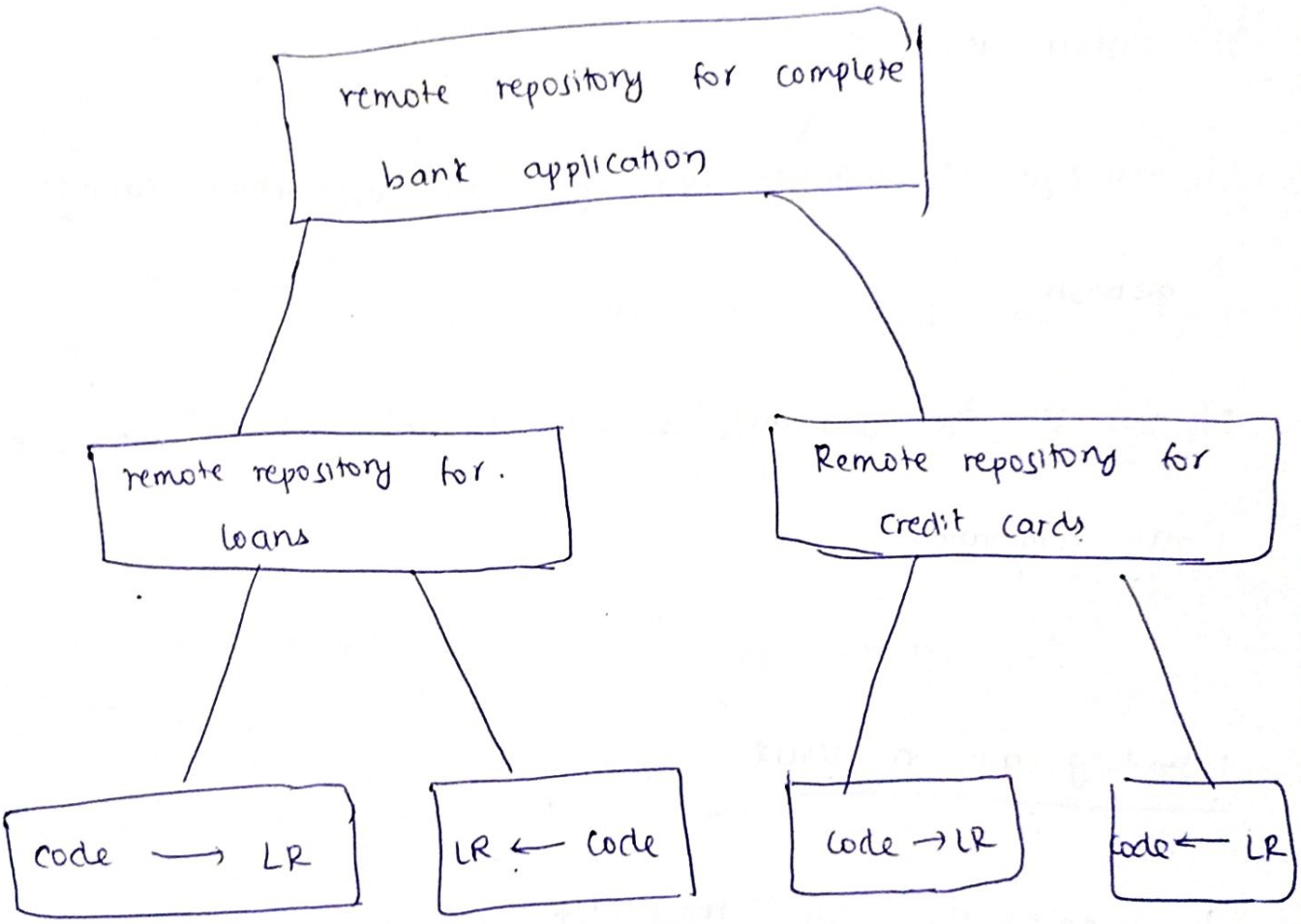
2. Distributed version controlling

In DVC. we have a local repository installed on every team member machines where version controlling happens at the level of individual team members later the code will pushed in to remote server where version controlling happens at level of entire team.



ex: GIT.

In DVC we can also maintain "bare repositories" i.e; repositories at the level of sub teams which can be merged at the level of complete project.



Installing Git on windows

1) open https://git-scm.com/download

2) Download git for windows

3) Install it.

4) once git is installed we get an application called gitbash

5) This is the command prompt of git where we fire git commands

Installing git on Linux

1. update the apt repository.

```
sudo apt-get update
```

2. install git

```
sudo apt get install -y git
```

Redhat / centos / OEL (oracle Enterprise linux)

1. update the yum repository & install git.

```
yum -y install git
```

Setting up username & email id for all users on machine

```
$ git config --global user.name "sai krishna"
$ git config --global user.email "udayalwala3@gmail.com"
```

To see the list of all global configurations

```
git config --global --list.
```

Git when working on the local machine uses three components.

1. working directory
2. Stagging Area
3. local repository

working directory or workspace is the folder where developer creates the code. Initially all the files here are called untracked files

Stagging Area is the intermediate buffer zone of git. It is also known as index area of git. files present here are called stagged files or indexed files

local repository is the location where version controlling happens on the local machine and the files here are called commited files.

1. To initialize the working directory as a git repository

git init

2. To send a file from working directory to stagging area.

git add filename.

3) To send multiple files into stagging area.

git add file1 file2 file3

to send all the files & folders in to stagging area

git add .

3) To bring files back from stagging to untracked section

git rm --cached filename.

(or)

git reset filename

To send files from staging to local repository

git commit -m "some message";

5) To see the stages of files in working directory and staggring area

git status.

6) To see the commits done in local repository

git log.

To see the commit history in one line format.

git log --oneline. (To see commit id & commit message)

.gitignore

This is the special configuration file which is used by git for storing private files info. Any file whose name is present in .gitignore cannot be accessed by git.

1. Create few files in the working directory

   touch f1 f2 f3 f4 f5.

2) Check the status of git it will show the above 4 files as untracked files

   git status

3) create .gitignore file and save the names of the above four files in it

   cat > .gitignore

   f1
   f2
   f3
   f4.

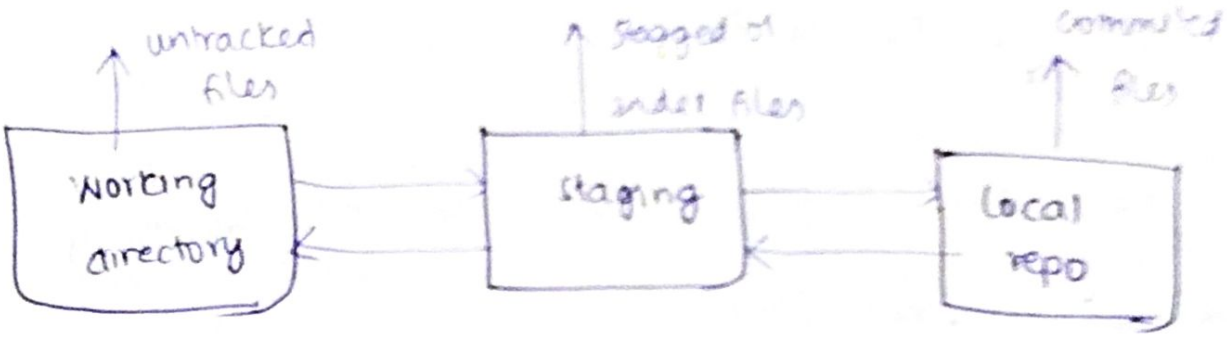
to come out of .cat command ctrl+d (EOF)

check the status of git

git status.

It will show only .gitignore as an untracked file

fi- fy will no longer accessed to git



GIT Branching

Branching is the feature of GIT which allows developers to create multiple functionalities code on seperate Branches and later this can be merged to master Branch. This helps the developers in creating the code in an uncluted ways.

1. To see the list of all local branches.

```
git branch.
```

To see the list of all branches (local & remote)

```
git branch -a.
```

2. To create a new Branch.

```
git branch branch-name.
```

3. To move in to a branch.

```
git checkout branch-name
```

4. To create branch and also move into it

git checkout -b branch-name.

5. To merge a branch

git branch branch-name

6. To delete a branch that is merged

git branch -d branch-name

(Soft-delete)

7. To delete a branch that is not merged

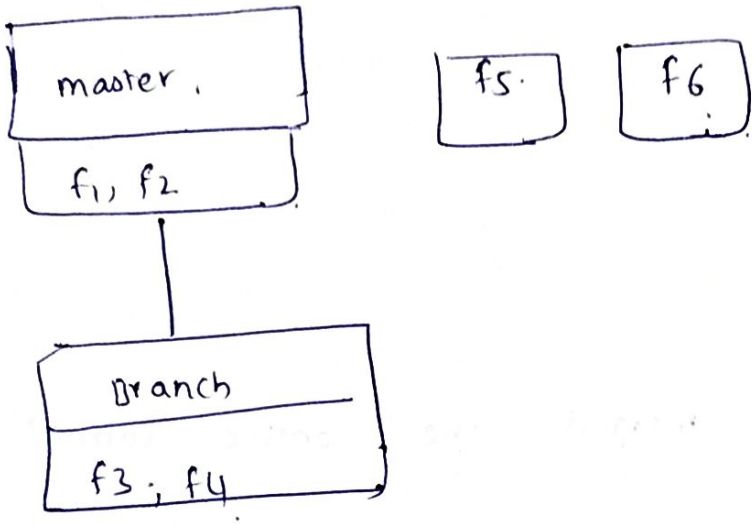
git. branch -D branch-name

This is hard delele

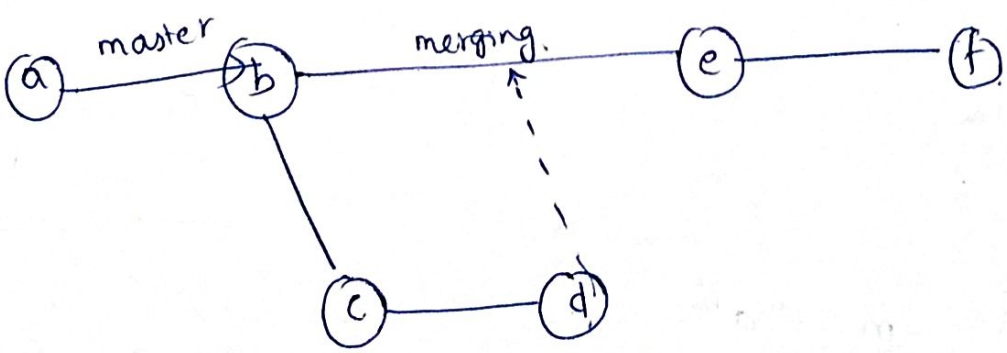
Note: whenever a branch is created the entire commit history of the master branch upto that point will be copied to newly created branch

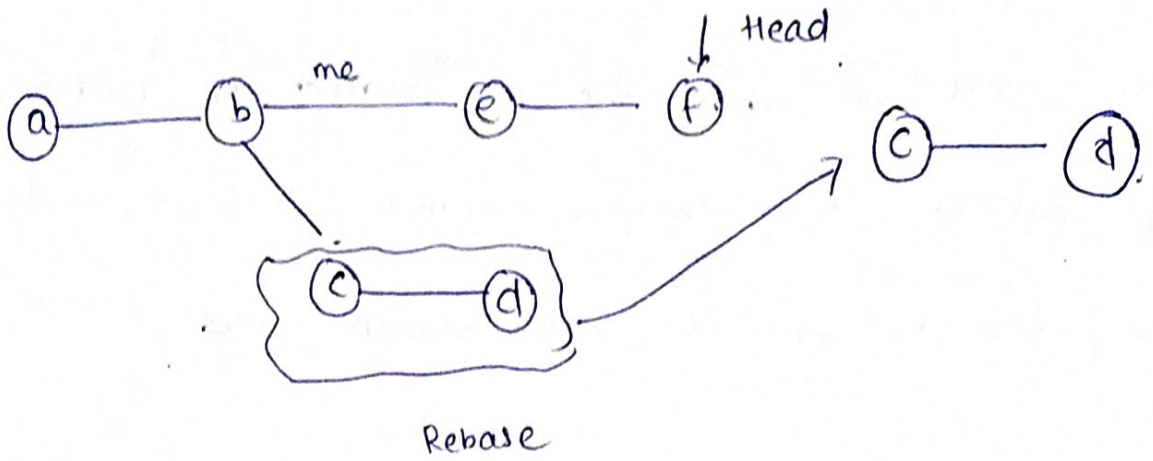
Note:

Irrespective of where file is created or modified, git only consider from which branch it is commited and that file belongs to that branch only



git merge test.





Rebase

Git merging

Whenever a branch is merged the entire commit history of that branch will be moved into the master branch based on the time stamps of the commits

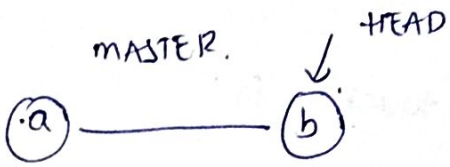
1. create few commits on master.

```
touch f1.

git add .

git commit -m "a".
```

touch f2

git add .

git commit -m "b"



2. Check the commit history

git log -online.

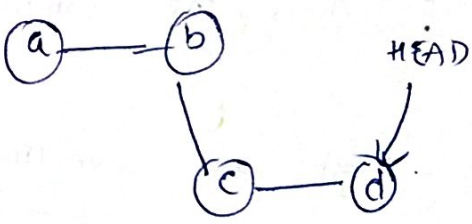
3. Git a new branch "test" and create few commits on it.

git checkout -b "test"

touch f3

git add .

git commit -m "c"



touch f4

git add .

git commit -m "d".

4. Check commit history

git log -oneline

5. move back to master and create more commits

git checkout master

touch f5.

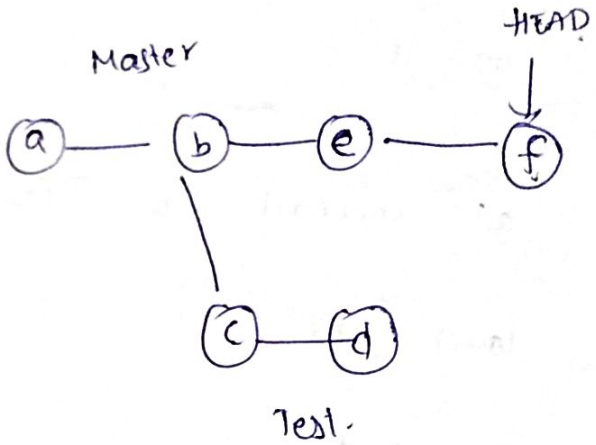
git add .

git commit -m "e"

touch f6

git add .

git commit -m "f".



6. check the commit history

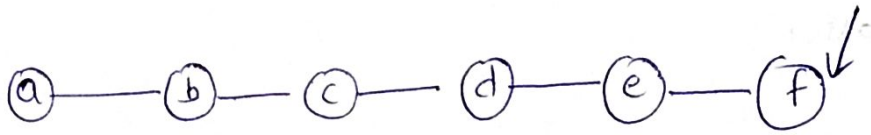
git log -oneline.

7. merge test with master.

git merge test.

8. check the commit history

git log --oneline



Git Rebasing

This is also known as fast forward merge. The commits coming from a branch will be projected as the top most commit (Head commits) on the master Branch.

Implement first 6 steps from previous scenario

7. To rebase test with master.

git checkout test.

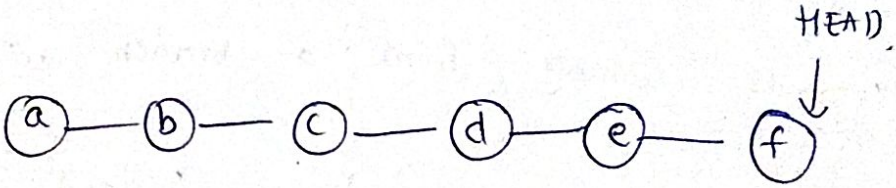
git rebase master.

git checkout master.

git merge test.

8) check the commit history

git log --oneline



Working on remote GIT repository

1) open github.com signup for a free account

2) Signin into that account

3) To upload the code from the local repository into the remote repository

Click on + icon on top right corner, click on new repository, enter some repository name select public (or) private click on create repository

Go to push an existing repository from the command line. copy the first command & paste it in git bash. this will create a link between local & remote repository

Copy the second command & paste in git bash.

git push -u origin master.

enter user name & pwd of github

All the code from local repository will be uploaded into the remote repository. This is also called as checkin.

downloading from the remote Git hub

This is also called as checkout & it can be done in 3 ways

1) Git clone
2) Git fetch
3) Git pull.

Git clone

This will download the code from the remote gitHub irrespective of whether that is code is already present in local machine or not. This was generally used for the first time when all the team members

want to download complete project

git clone remot-git-repository-url.

Git fetch

This will work only when the code present on remote server is different from the code present on the local machine

It will download the modified files and place them on a seperate branch which is called the remote branch we can go in to this remote branch check if the modifications are acceptable and if they are acceptable merge it with master Branch

1) Sign into github.com.

2) Click on the repository that we uploaded

3) Click on the file that we want to modify

Click on edit icon

4) make some changes → click on commit changes

5) open gitbash.

6) git fetch.

7) to see the list of all branches

git branch -a.

8) move into the remote branch

git checkout remotes/origin/master

9) view the changes & if the changes are acceptable

merge with master.

git checkout master

git merge remote/origin/master.

Git pull

This will also work only when the code present on local machine is different from code present on the remote server. but it will directly download the code and merge it with the master branch

6) git pull

7) The modified files will be directly available on the master branch

Git cherry pick

This is the feature of git which allows us to selectively pickup only certain commits from a branch and add them to the master branch.

Create few commits on master.

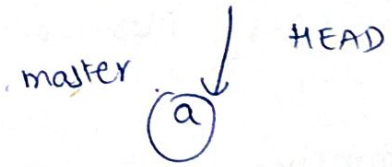
touch f1.

git add .

git commit -m "a"

2. Check the commit history of master.

git log --oneline



3. create a branch "test" and create few commits on it.

git checkout -b test

touch f2.

git add .

git commit -m "b".

touch f3

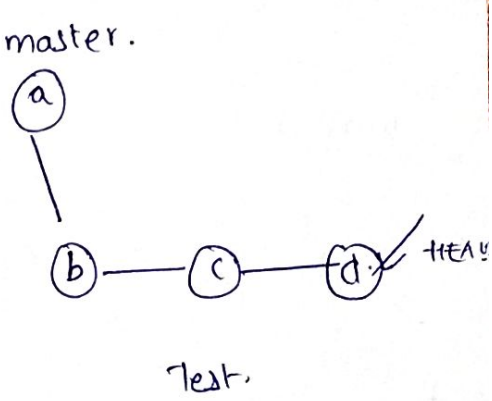
git add .

git commit -m "c"

touch f4

git add .

git commit -m "d"



4) check the commit history of test

git log --oneline.

5) Move to master & cherry pick on the required commits

git checkout master

git cherry-pick commit-id1 commit~~id~~-id3.

* Irrespective of where the file is created or modified * git only consider from which branch the file is commited the file. is belong to the commited branch only

git m-

master

touch f1 f2 f3 f4

git add .

branch 1

touch f5 f6

git add .

branch 2

git commit -m "message"

So the files belongs to the branch 2 only

Git Reset

This is classified into three types

1) Hard reset
2) soft reset
3) mixed resel.

Hard reset :-

This will make the head point to an older commit and we can see the data as it was present at the time of older commit.

1) create a file file1 and store "one".
2. send it to stagging and commit.

```
git add .
git commit -m "a".
```

3. open the same file file1 and store

```
one
two
```

```
git add .
git commit -m "b"
```

4) open the file1 again and store

```
one
two
three
```

git add .

git commit -m "c"

5) check the commit history

git log --oneline

6) perform a hard reset to "b" commit.

git reset --hard "b-commit-id"

7) open file1 we see content as

one
two.

and the HEAD is pointing to b commit.

Soft reset

this will also move the head pointer to older commit but the files will move one stage back before the commit operation took place i.e, they will move into the staging area

git reset --soft "b-commit-id"

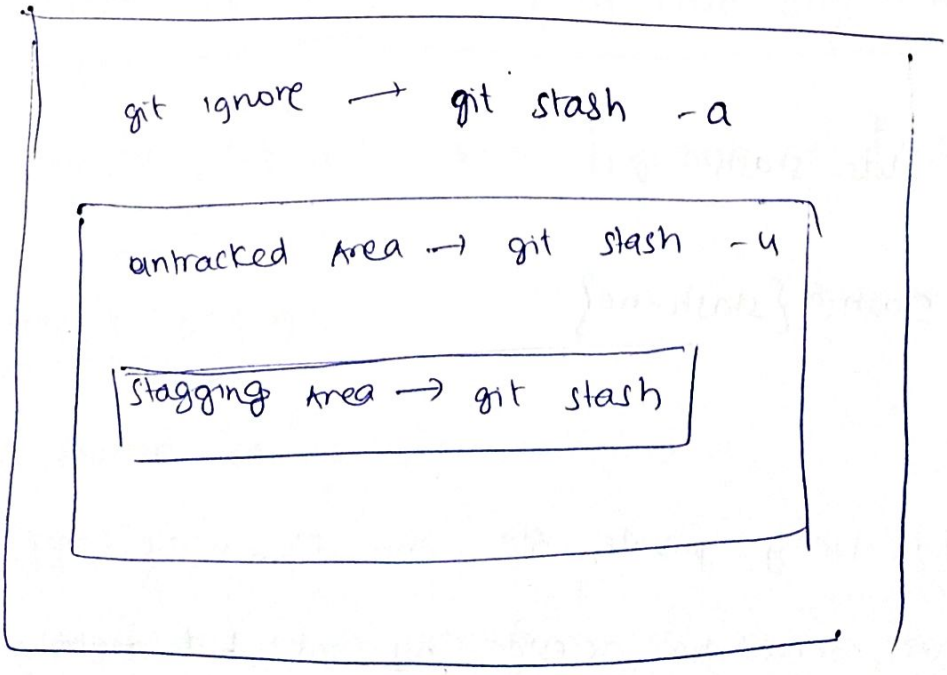
mixed reset

This will also make the HEAD point to an older commit but the status of git will move 2 steps back. i.e; the folder files will be displayed in the untracked / modified section.

```
git reset --mixed "b-commit-id"
```

git stash

This is the feature of git which is used for moving files into the stash section of git. Any files that is moved into the stash section cannot be accessed by git until it is unstashed.



stash all the files in the stagging area.

git stash.

2) to stash all the files in the stagging area and untracked section

git stash -u

3) to stash all the files in stagging, untracked, .gitignore

git stash -a.

4) To see the list of all the stashes

git stash list

5) to unstash the latest stash

git stash pop

6) To unstash an older stash.

git stash pop stash@{stash-no}.

Note:

.gitignore is used for hiding private file. Any file whose name is mentioned in .gitignore cannot be accessed by git. but ~~.gitol~~ .gitignore itself is accessed by git i.e; it will move into staging area, local repo and the remote repository

To hide .gitignore also we can use

git stash -a

Adding origin (remote)

git remote add origin https://github.com/udayalwala/uday.git

account name

repository name

To push in to the origin master

git push -u origin master.

To show remote origin

git remote show origin

To change the origin (in case origin already exists)

git remote set-url https://github.com/udayalwala/uday1.

To remove origin

git remote rm origin.

(or)

git remote remove origin