

What is Git?

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

It is used for:

* Tracking code changes
* Tracking who made changes
* Coding collaboration

What does Git do?

* Manage projects with **Repositories**
* **Clone** a project to work on a local copy
* Control and track changes with **Staging** and **Committing**
* **Branch** and **Merge** to allow for work on different parts and versions of a project
* **Pull** the latest version of the project to a local copy
* **Push** local updates to the main project

Working with Git

* Initialize Git on a folder, making it a **Repository**
* Git now creates a hidden folder to keep track of changes in that folder
* When a file is changed, added or deleted, it is considered **modified**
* You select the modified files you want to **Stage**
* The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files
* Git allows you to see the full history of every commit.
* You can revert back to any previous commit.
* Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

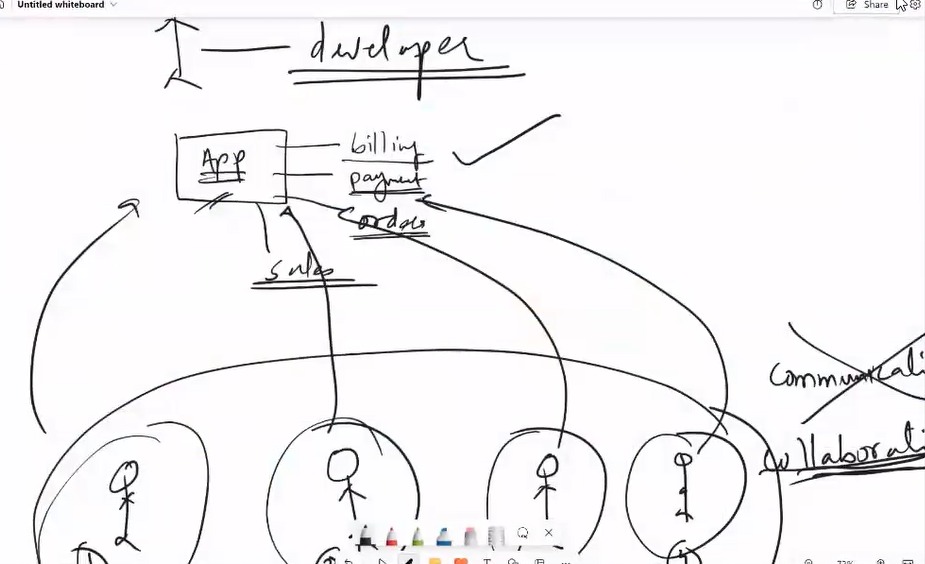
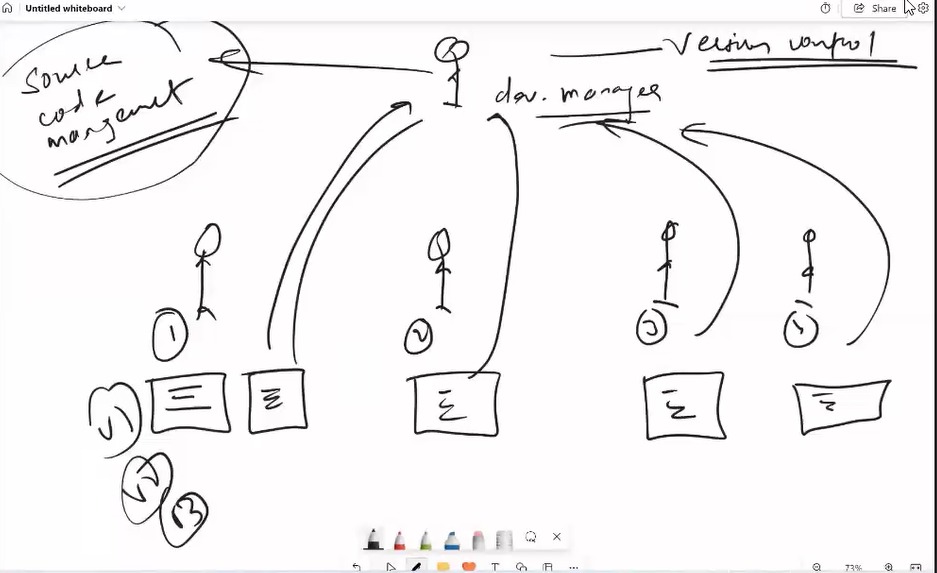
Why Git?

* Over 70% of developers use Git!
* Developers can work together from anywhere in the world.
* Developers can see the full history of the project.
* Developers can revert to earlier versions of a project.

What is GitHub?

* Git is not the same as GitHub.
* GitHub makes tools that use Git.
* GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.
* In this tutorial, we will focus on using Git with GitHub.

What is the Difference Between Distributed and Centralized Version Control Systems



**What is Version Control**

A version control system (VCS) is the software that records changes made to source code, including separate files, sets of files, and any digital assets and related project metadata. This way, the system enables tracking of all project changes and allows developers to revert files to specific versions as needed. Without version control, projects risk becoming disorganized with numerous file versions.

Also, VCS lets all team members work simultaneously on the latest file version for a specific project. This ensures faster and more efficient product development and delivery, crucial in the DevOps workflows. Moreover, VCS allows for conflict detection and resolution before issues reach production.

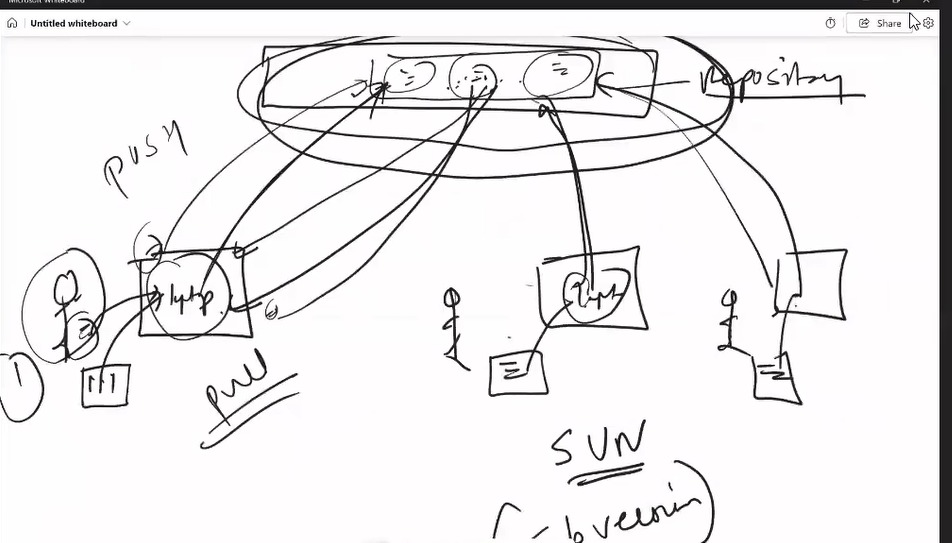
Overall, version control systems offer several benefits:

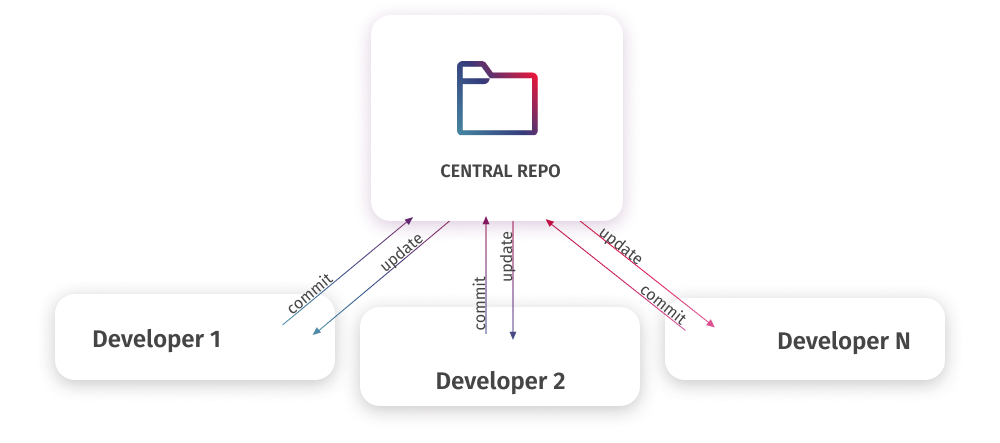
* **Version management**: VCS stores all versions of changes down to the smallest developer commit.
* **Version restoration**: Developers can restore specific versions to address conflicts or errors.
* **Security**: VCS provides role-based access control, preventing unauthorized code access.
* **Integrity**: Integration with other DevOps tools ensures stable processes.
* **Collaboration**: VCS facilitates collaborative teamwork with a focus on individual work goals.
* **Backup**: Modern VCS can serve as a backup for project codebases.

There are two main types of version control systems: Centralized Version Control Systems (CVCS) and Distributed Version Control Systems (DVCS). Let’s explore both.

**Centralized Version Control Systems**

A centralized version control system operates on a client-server model. In this setup, the central server hosts the master repository, which contains all versions of the code. Developers begin by pulling the latest source code version to their local machines to make modifications. Once changes are made, they commit these changes to the central repository, where conflicts are resolved, and the updates are merged.



[](https://blog.devart.com/wp-content/uploads/2024/05/Centralized.png)

In this model, only one developer can modify a specific piece of code at a time. The system locks the file to prevent other developers from accessing it while it is being edited. Team members may create branches to work independently, but all changes are eventually committed to the central repository. After merging, the server unlocks the files.

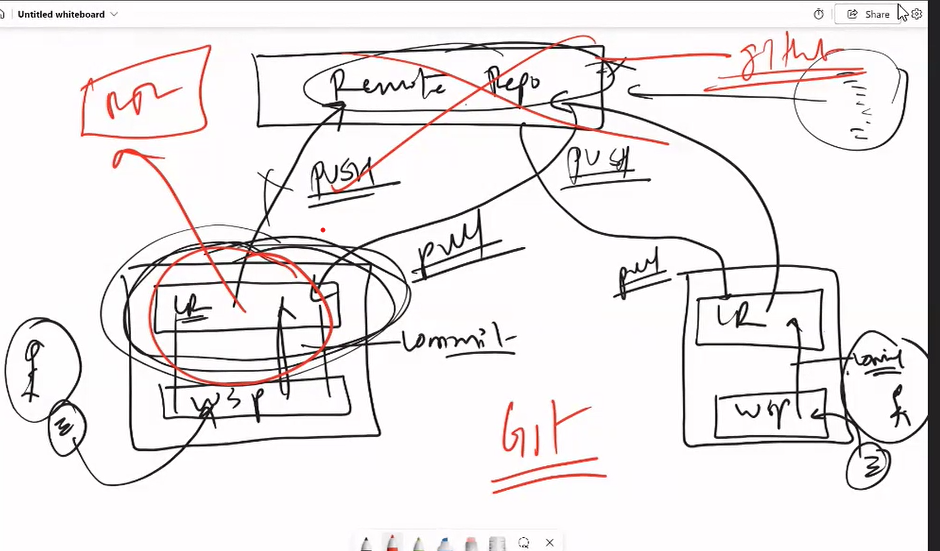
The centralized approach is best suited for smaller teams where direct communication and coordination are feasible, as this is vital for maintaining an effective workflow.

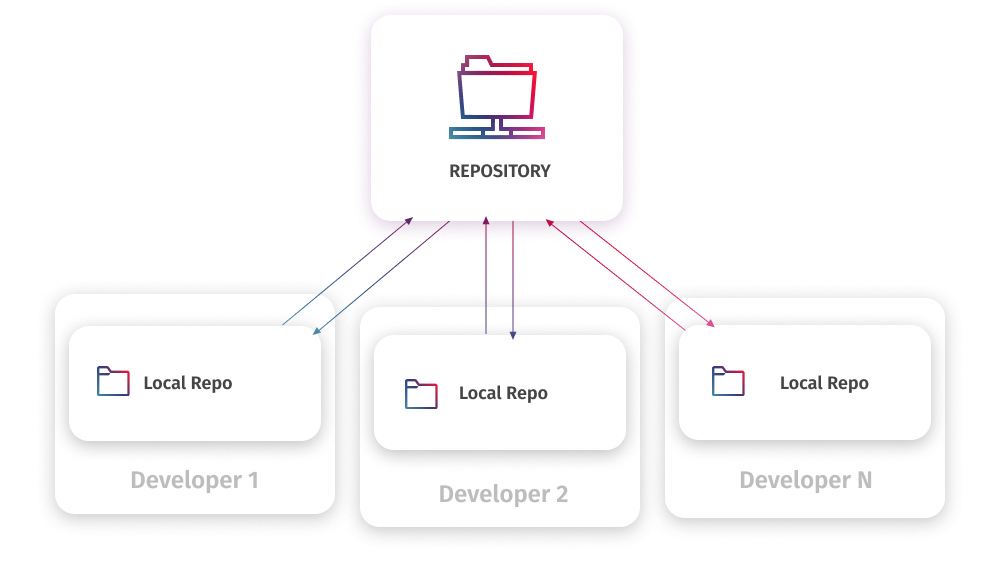
Subversion (SVN) is a widely used centralized version control system. Unlike other systems that support branching, SVN manages all project files in a single line. It simplifies scaling for large projects but requires comprehensive, robust security policies to control access to different areas of the project.

**Distributed Version Control Systems**

Distributed version control systems (DVCS) function similarly to centralized models in most aspects but with a significant difference. Instead of a single server holding a repository, each developer maintains their own repository on the local machine. That local repository contains the entire history and all branches.

In practice, using DVCS means that each user has a mirrored copy of the entire repository on their local machine. They can branch, commit, and merge changes locally without needing the server to store physical branch files; it only needs the commit differences.



[](https://blog.devart.com/wp-content/uploads/2024/05/Distributed.png)

After making changes, the developer commits them to the local repository. However, at this stage, the local repository is separate from the master repository, resulting in different sets of changes between the developer’s contribution and the master repository. Developers don’t directly merge their code into the master repository. Instead, they request to push these changes from their local copy to the master repository.

The main advantage of the distributed model is that such systems allow users to work independently, even without a direct connection to the central repository. Therefore, even a failure in the central repository won’t affect local work. Besides, with code review processes in place, only clean, high-quality code can be merged into the main repository.

Although DVCS can be complex, especially for new developers, the benefits of the distributed model justify investing time and effort in mastering these systems. Multiple developers can collaborate efficiently and deliver excellent software.

The most widely used examples of DVCS are Git and Mercurial.

Git stands out as the most popular version control system overall. It’s an open-source DVCS suitable for projects of any size and complexity, widely used in startups and enterprises alike.

Mercurial is another DVCS with straightforward branching and merging features, ideal for scalable projects, and it offers an intuitive user interface. This visual mode makes it easy for even new users to quickly grasp the functionalities and work efficiently.

**Centralized vs Distributed**

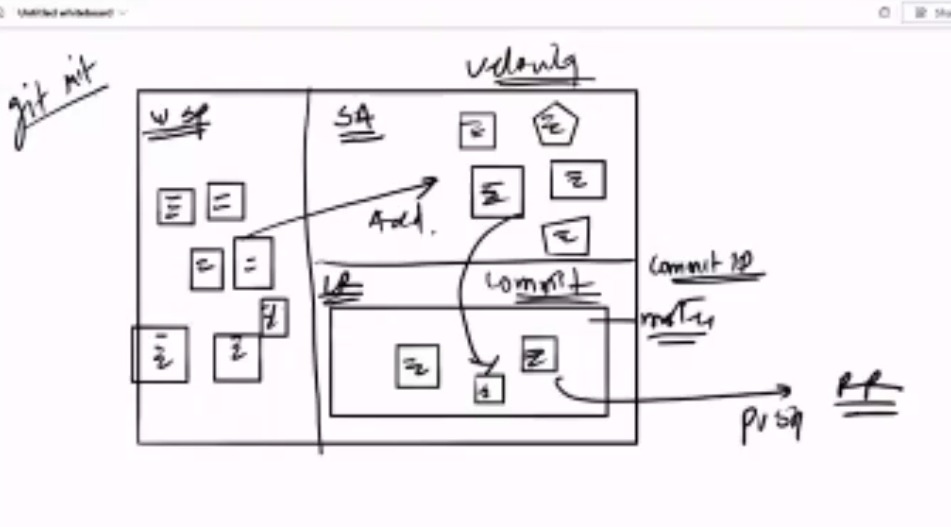
Let us review the specific features of both systems in the comparison table:

|  |  |  |
| --- | --- | --- |
|  | **Centralized** | **Distributed** |
| **Repository** | Single central repository | Separate cloned repositories for every user |
| **Connection** | Requires constant connection to the central repository | Does not require constant connection, can work offline |
| **Branching and merging** | Limited | Extensive |
| **History** | The central repository contains the version history | Each local repository has a full version history |
| **Access control** | Permissions are set at the central server | Permissions are set at both local and remote repositories |
| **Speed** | Slower performance as operations rely on the central server | Faster performance as most tasks are done locally |
| **Data loss risk** | Failures on the central server can lead to data loss, and comprehensive backups are required | A full copy of the repository is available to each user, reducing data loss risks |
| **Collaboration** | Requires constant coordination with the central server | Allows concurrent work and less centralized control |
| **Adaptability** | More rigid | More adaptable |

## GitHub Account

Go to [GitHub](https://www.github.com/) and sign up for an account:

## Initialize Git



## Git init command initialize your current folder as the local repository

Once you have navigated to the correct folder, you can initialize Git on that folder:

### Example

git init

Initialized empty Git repository in /Users/user/myproject/.git/

You just created your first Git Repository! **Note:** Git now knows that it should watch the folder you initiated it on.

Git creates a hidden folder to keep track

yum install git -y ----- to install git on linux

git -v ---- to check git version

git init ----- initialize the git repo

when we initialize the git repo is divided into 3 parts

1. wsp/working dir ----- here we edit and create files/folders

2. Staging Area ---- we finiailize our change/we track the change here

3. Local Repository- ----- this is where we actually commit our changes to further push them to Remote repo i.e github

**in LR there is a default branch called as master created. think of this as a logical folder as of now.**

git status ---- shows the status of git repo whether it is wsp or SA

git add filename ---- adds the file/change to the SA

git rm --cached index.html ---- again went inside the wsp(revert the changes)

role of devops engineer

1—create remote repo

2—access control

commit—means changes which we are doing in LR

git commit -m "message" --- commits all the changes present in SA to local repository

git log ---- shows all the commits

**Git config cmd**

git config --global user.name username ----- to set username on the machine

git config --global user.email useremail ----- to set useremail on the machine

Change the user name and e-mail address to your own. You will probably also want to use this when registering to GitHub later on.

**Note:** Use global to set the username and e-mail for **every repository** on your computer.

If you want to set the username/e-mail for just the current repo, you can remove global

git config --list ---- to check the user config

git config --global --edit ------- to edit the config

HEAD ---- head is the pointer to the latest commit

git show commitID ----- shows details of the commit.track of changes.

**Cmd to push changes from local master to remote repo**

git remote add origin url ---- to set remote origin

git remote -v

git remote show origin ---- to see the set origin

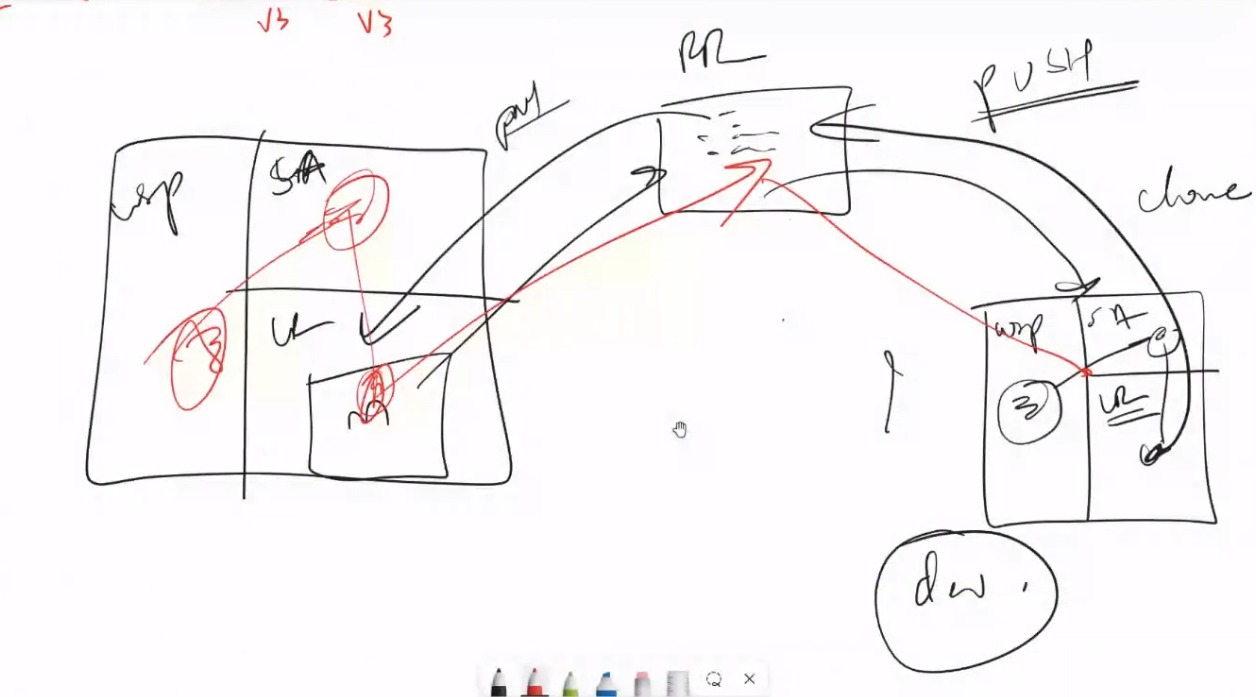
git remote rm origin ---- to delete the origin url

git push origin master ---- to push the local repo changes to the RR

git clone URL --- to download a copy of remote repository locally with commit history

git pull origin master ----- to download the change in the RR locally

**Scenario one- if first developer commit changes in remote repo then developer 2 wants to do the changes in the same repo then developer2 needs to clone the remote repo first (clone command download the copy of remote repo in the developer 2 LR)then made the changes and then commit and suppose developer1 wants to do the some changes then he needs to pull the changes first and then commit.**



**Interview specific cmd**

*To change the latest commit message we can use the –amend cmd*

*Cmd: git commit –amend -m “new message”*

*Change Auther:*

*Cmd: git commit –amend –author “velocity<*[*velocity@gmail.com*](mailto:velocity@gmail.com)*>”*

**How we can manage the folder using git**

**Steps:**

*Create a folder << mkdir aws*

*Create one file inside the folder<< index.html*

*Create empty folder << mkdir velocity*

*Git status<< only aws folderfiles will be displayed as* ***Git does not recognize the empty folder/dir***

***.gitignore***

***If we have one empty folder still we want to push the empty folder then we need to create .gitignore file inside the empty folder and then push***

*Mkdir ec2*

*Cd ec2*

*touch .gitignore*

*Git push origin master*

***If we want to ignore some file which is wsp***

*Touch index.html new.jsp index.jave file1.txt file2.java new.html file3.html*

*Vi .gitignore*

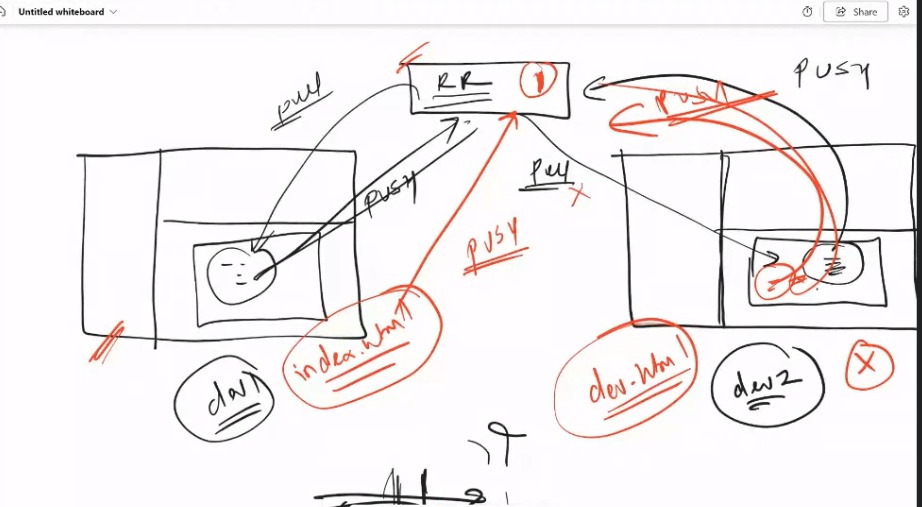
*Add << \*.java*

*\*.html*

*Java and html file will be ignored from wsp*

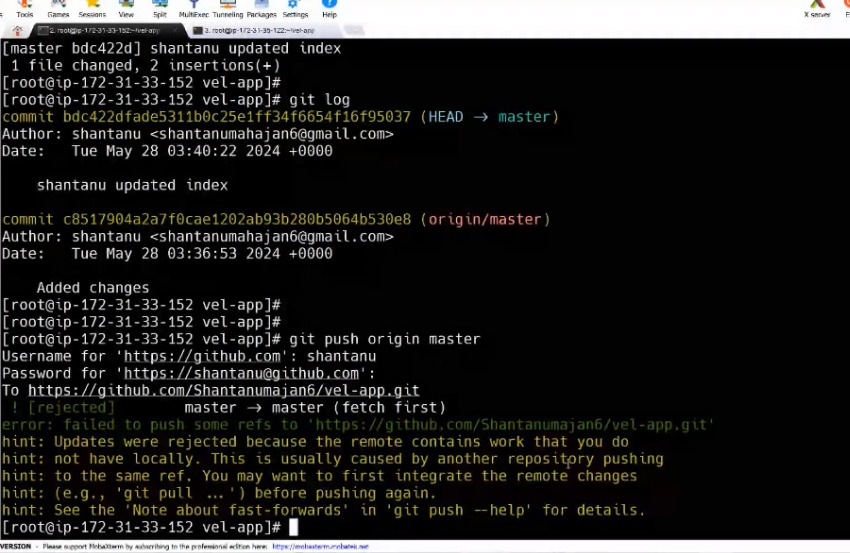
***Merge:***

***(changes in different file)***



*we have a two developer dev1&dev2*

*dev1 making the changes in index.html and push the changes on Remote Repo and simultaneously dev2 also making the changes in dev.html and pushing the changes on RR without pulling the changes done by dev1 then there will be the conflict*



*To resolve this error we need to merge the changes first.*

*Pull the changes first then merge and then push the changes*

***Commands:***

*25 yum install git -y*

*126 clear*

*127 cd*

*128 clear*

*129 git config --global user.name shantanu*

*130 git config --global user.email shantanumahajan6@gmail.com*

*131 cd /mnt*

*132 mkdir vel-app*

*133 cd vel-app/*

*134 ls -ltr*

*135 git init*

*136 vi index.html*

*137 vi dev.html*

*138 git add \**

*139 git commit -m "added 2 files"*

*140 git remote add origin https://github.com/Shantanumajan6/vel-app.git*

*141 git push origin master*

*142 git log*

*143 clear*

*144 git pull origin master*

*145 ls -ltr*

*146 git log*

*147 vi dev.html*

*148 git add dev.html*

*149 git commit -m "added dev1 change"*

*150 git log*

*151 git push origin master*

*152 git log*

*153 git pull origin master*

*154 git config pull.ff true*

*155 git pull origin master*

*156 git log*

*157 git log --oneline*

*158 git push origin master*

*159 git log --oneline*

*160 clear*

*161 ls -ltr*

*162 vi index.html*

*163 git add index.html*

*164 git commit -m "added change by shantanu"*

*165 git log -3*

*166 git push origin master*

*167 git pull origin master*

*168 git log*

*169 git push origin master*

*170 git log --oneline*

*171 cd ../*

*172 ls -ltr*

*173 rm -rf \**

*174 clear*

*175 mkdir velocity-app*

*176 cd velocity-app/*

*177 git init*

*178 vi index.html*

*179 vi dev.html*

*180 git add \**

*181 git commit -m "added 2 files"*

*182 git remote add origin https://github.com/Shantanumajan6/velocity-app.git*

*183 git push origin master*

*184 ls -ltr*

***191 git push origin master***

***192 git pull origin master***

***193 git config pull.ff true***

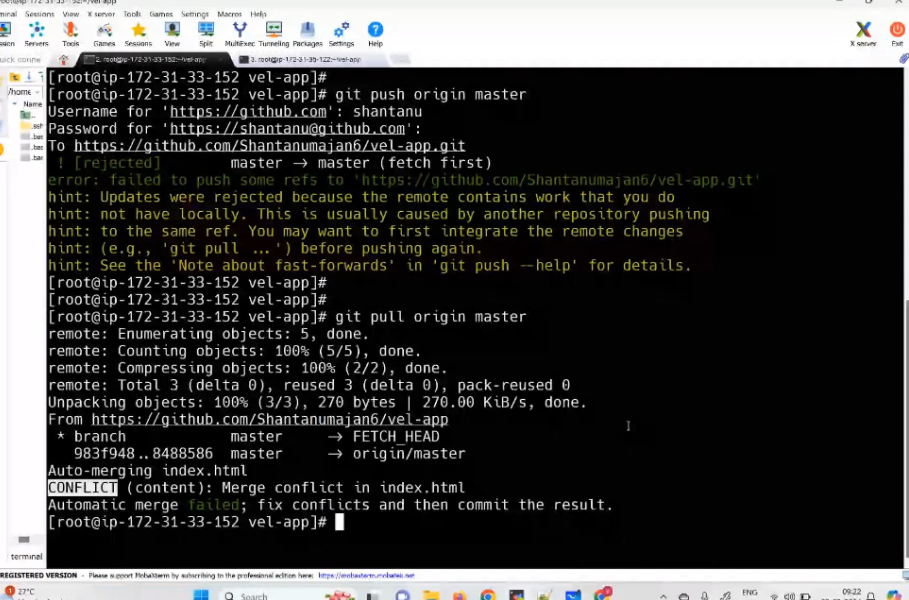
***194 git pull origin master***

***Then merge master vi editor will open we need to only save and close the editor. Extra merge commit will be added and changes from origin master and LR will be merged***

*198 git push origin master*

*199 git log –oneline*

*If both developers doing changes on same file ex:dev1 change the index.html file and update x=10 and try to push and without pull dev2 update the index.html as x=100 and try to push then there will be below conflict*

**

*How to resolve merge conflict:*

*185 vi index.html*

*186 git add index.html*

*187 git commit -m "Added x=10 by shantanu"*

*188 git log*

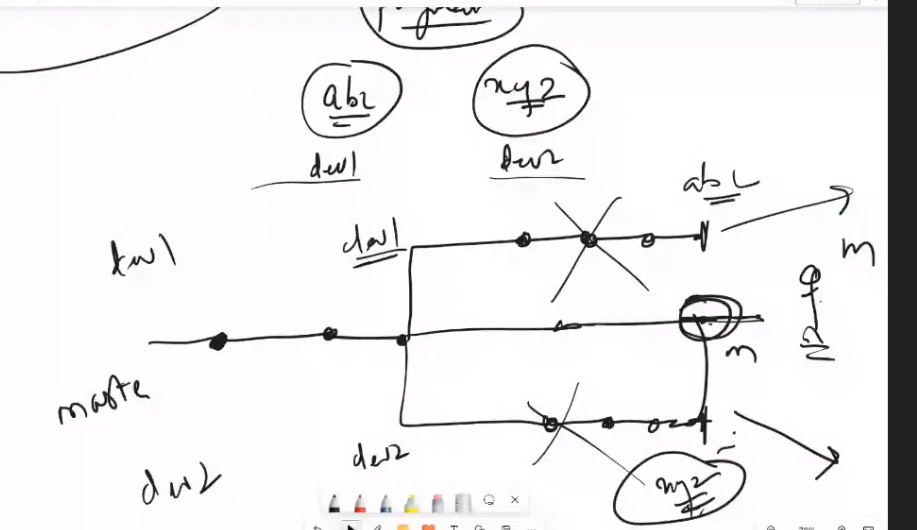
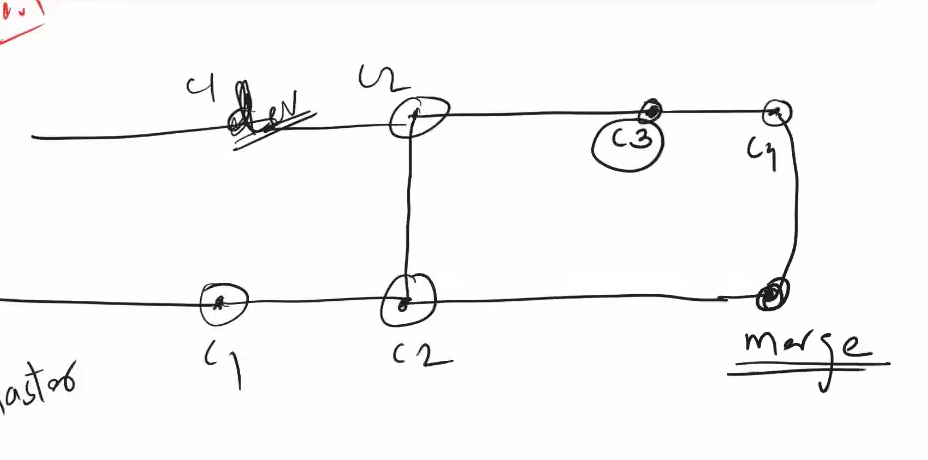
*189 git commit --amend -m "added x=100 by shantanu"*

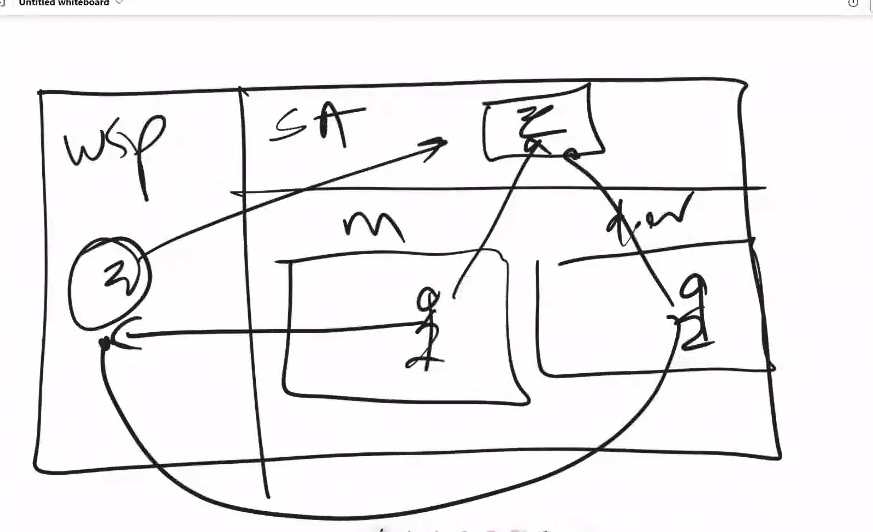
*190 git log*

***Branching:***

***Individual line of development***

***Branch create from another branch and from latest commit(head) whatever previous commit was there it will also add in that branch and then those branch merge with master then we can delete the branches which we were created.***

*** ***

******

*2 clear*

*3 yum install git -y*

*4 git config --global user.name shantanu*

*5 git config --global user.email shantanumahajan6@gmail.com*

*6 clear*

*7 mkdir velocity-app*

*8 cd velocity-app/*

*9 clear*

*10 git init*

*11 ls -ltr*

*12 clear*

*13 vi index.html*

*14 git add index.html*

*15 git status*

*16 git commit -m "Added index"*

*17 git log*

*18 git branch*

*19 vi dev.html*

*20 git add dev.html*

*21 git commit -m "Added dev file"*

*22 git log*

*23 git branch*

*24 git branch dev*

*25 git branch*

*26 git checkout dev*

*27 git branch*

*28 git log*

*29 git checkout master*

*30 vi test.yaml*

*31 git branch*

*32 git status*

*33 git checkout dev*

*34 git branch*

*35 git status*

*36 git branch*

*37 git add test.yaml*

*38 git status*

*39 git checkout master*

*40 git status*

*41 git checkout dev*

*42 git branch*

*43 git commit -m "added yaml file to dev branch"*

*44 ls -ltr*

*45 git log*

*46 git checkout master*

*47 git log*

*48 clear*

*49 git branch*

*50 ls -ltr*

*51 git checkout dev*

*52 ls -ltr*

*53 git branch*

*54 vi qa.html*

*55 vi index.html*

*56 vi test.yaml*

*57 git status*

*58 git add \**

*59 git commit -m "Added changes to dev"*

*60 git log*

*61 ls -ltr*

*62 git branch master*

*63 ls -ltr*

*64 git checkout master*

*65 ls -ltr*

*66 git lo*

*67 git log*

*68 git branch*

*69 git log*

*70 git merge dev*

*71 git log*

*72 clear*

*73 git branch*

*74 git checkout dev*

*75 git branch*

*76 git checkout -b qa*

*77 git branch*

*78 vi qa.yaml*

*79 git add qa.yaml*

*80 git commit -m "added yaml file in qa"*

*81 git log*

*82 vi index.html*

*83 vi dev.html*

*84 git add \**

*85 git commit -m "added"*

*86 git log -4*

*87 git checkout dev*

*88 git merge qa*

*89 git checkout master*

*90 git merge qa*

*91 git log*

*92 git remote add origin https://github.com/Shantanumajan6/velocity-app.git*

*93 git push origin master*

*94 git push origin dev*

*95 git push origin qa*

*96 cd ../*

*97 rm -rf velocity-app/*

*98 clear*

*99 ls -ltr*

*100 git clone https://github.com/Shantanumajan6/velocity-app.git*

*101 ls -tlr*

*102 cd velocity-app/*

*103 ls -ltr*

*104 git branch*

*105 git pull origin dev*

*106 git branch*

*107 cd ../*

*108 ls -ltr*

*109 rm -rf velocity-app/*

*110 clear*

*111 git clone https://github.com/Shantanumajan6/velocity-app.git -b dev*

*112 ls -ltr*

*113 cd velocity-app/*

*114 ls -ltr*

*115 git branch*

*116 git pull origin dev*

*117 ls -ltr*

*118 clear*

*119 git branch*

*120 git branch shantanu*

*121 git log*

*122 clear*

*123 git branch*

*124 git branch -d shantanu*

*125 history*

*git branch -- we can see the current branch*

*git branch dev -- creates dev branch*

*git checkout -b dev -- creates branch and switches to newly created branch*

*git push origin dev*

*git push --all*

*git pull origin dev*

*git pull --all*

*git checkout dev --- to switch*

*git branch -d dev -- deletes branch*

*git branch -D dev --- forcefully deletes*

*#############################################################*

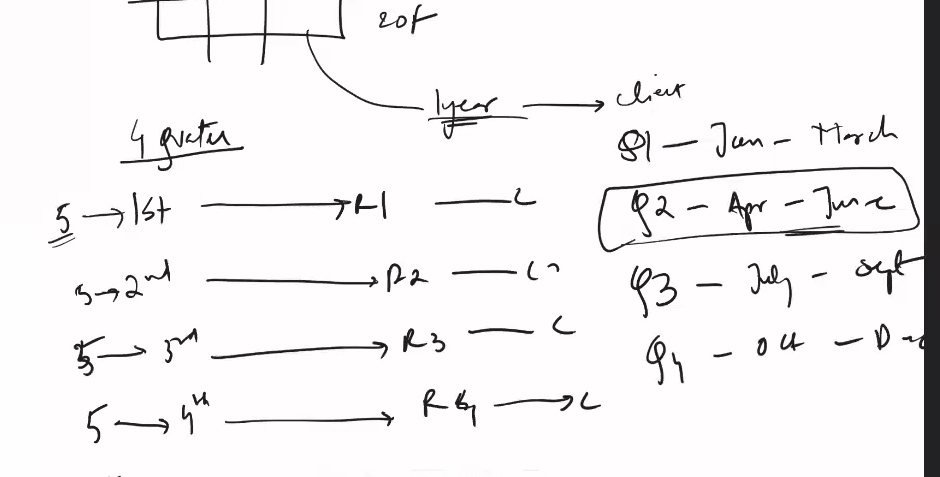
*git checkout master*

*git merge dev - -- here you should be in master here its merging dev branch into master branch*

*git clone URL -b branchname ---- to clone a particular branch, by default master branch will be cloned*

**Important:**

**Branching Statergies**

****

**New realease new branch: quarterly based**

**In Our project we are having quaterly based release.**

**for in every quater we have a main release and we create main release branches in every quater.**

**for for Q1 we have 2024Q1 branch for q2 release we have 2024Q2 branch which we created from 2024Q1 branch.**

**types of branch**

**1. Main release branch:-**

**this branch has commits which will go to production deployment after the release is completed.**

**2. Feature Branch:-**

**Feature branches are created when dev team is working on some feature, and feature branch is then merged in the Main release branch after verifiction.**

**if this feature is needed in previous releases then we can also back merge this feature in the pervious main release branches.**

**3. Bug Fix Branch:-**

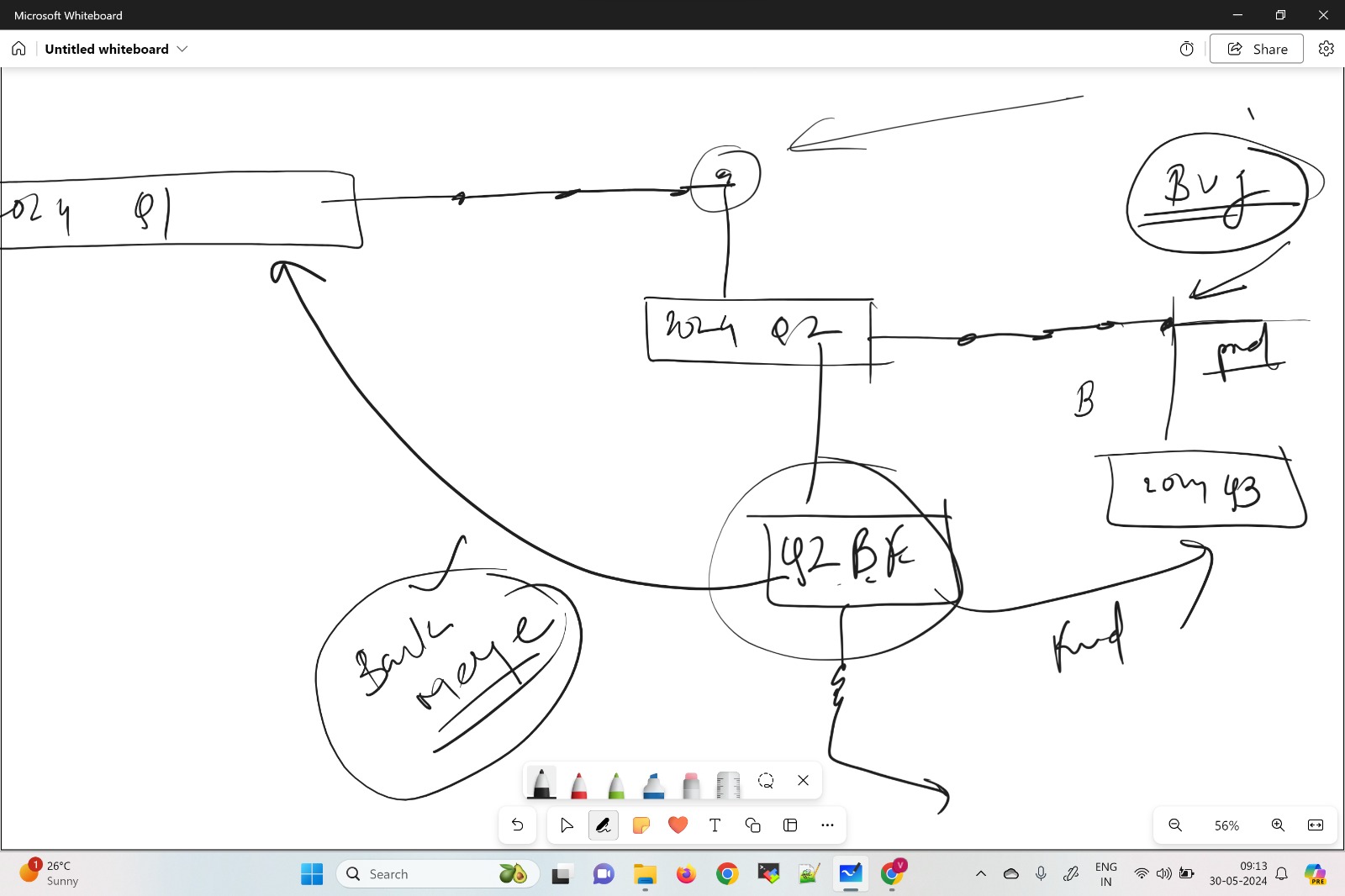
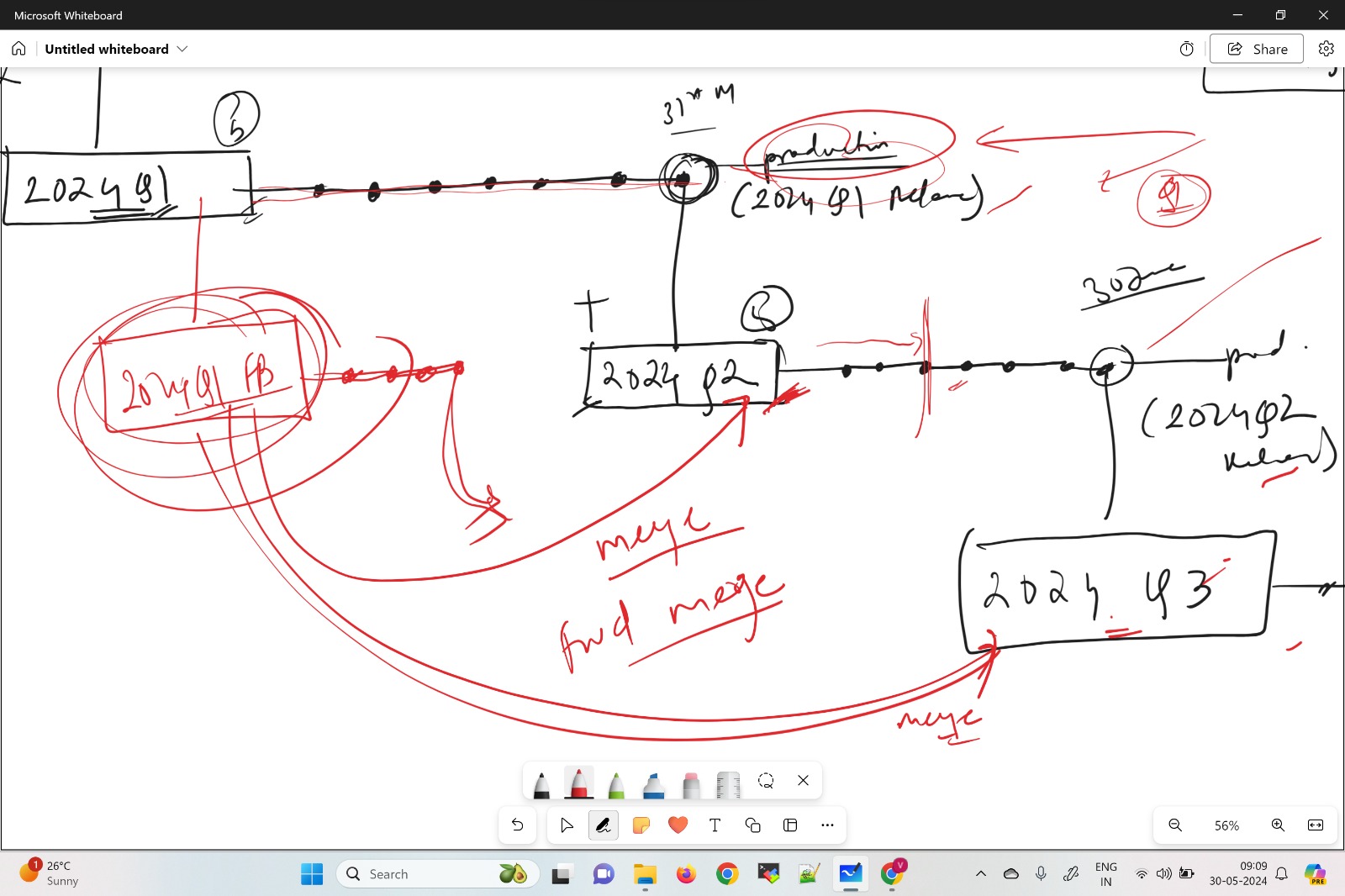
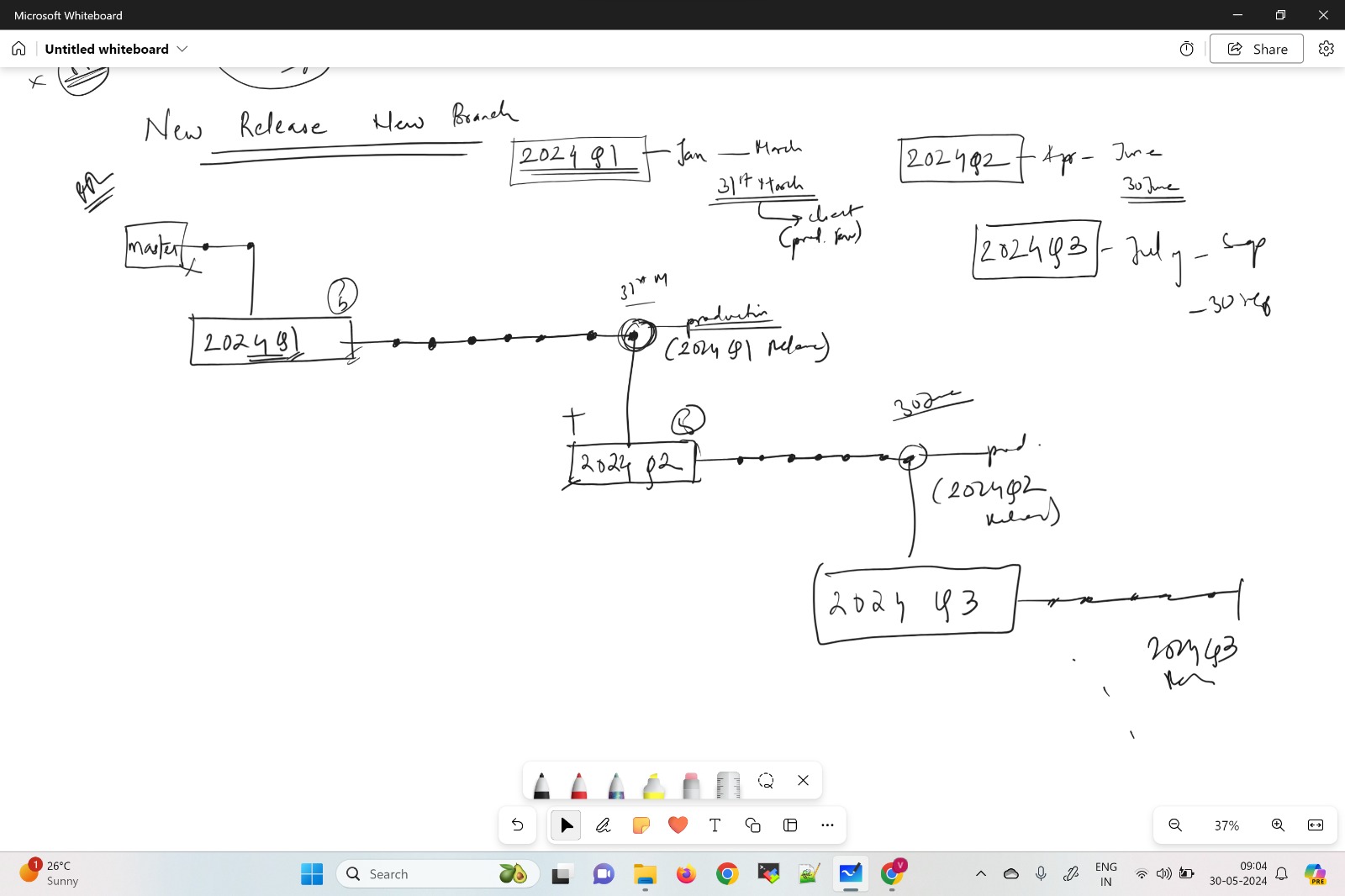
**when a client is having a release and has some issue or he finds some bug, we then create a bug fix branch from the**

**main release branch which is client is using.**

**after the fixes are done in the bug fix branch we provide it to the client.**

**if the same bug is found in the later releases then we can forward merge it in the further branches,**

**also can be back merged if required.**



Lets say we have a first release 2024-q1—

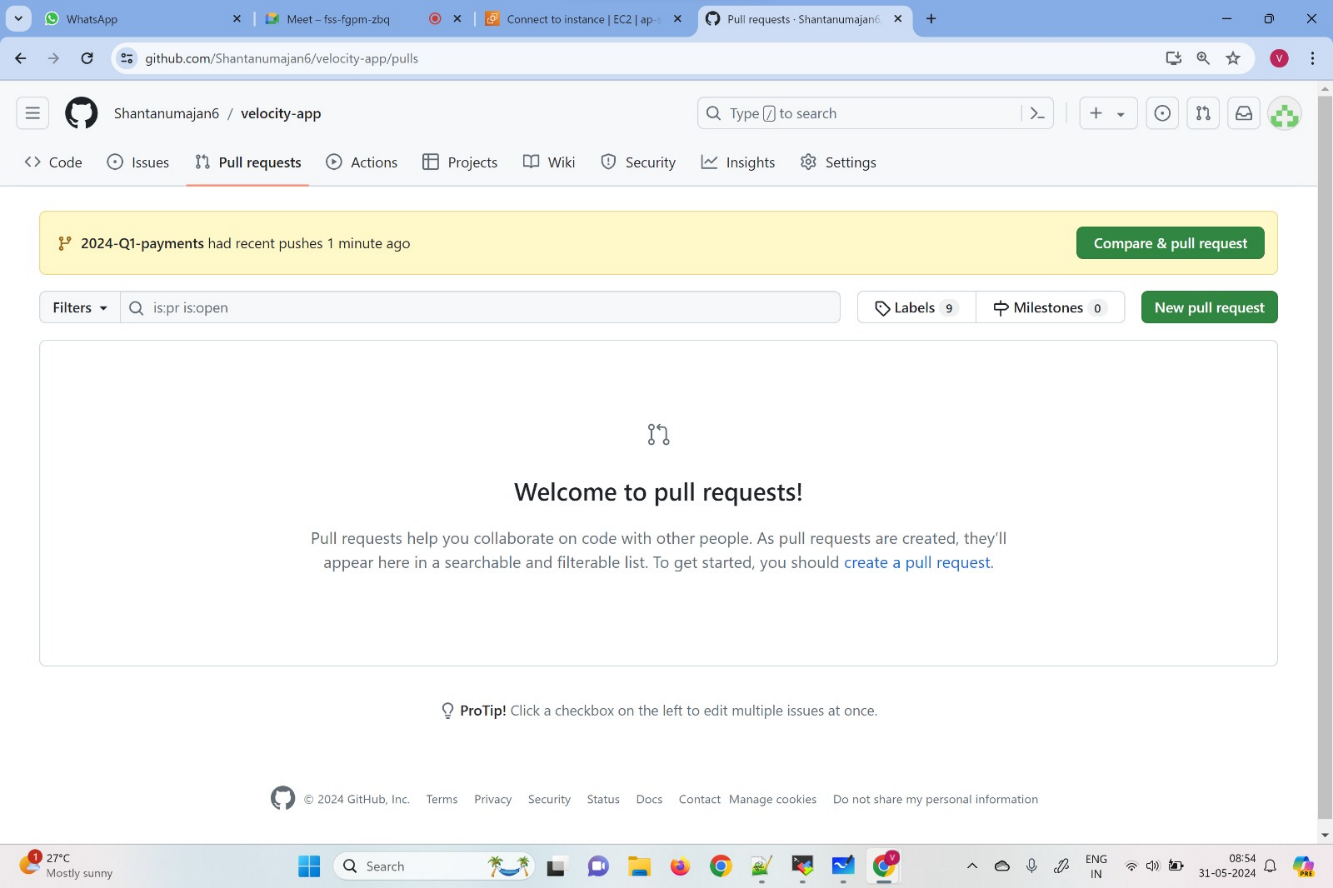
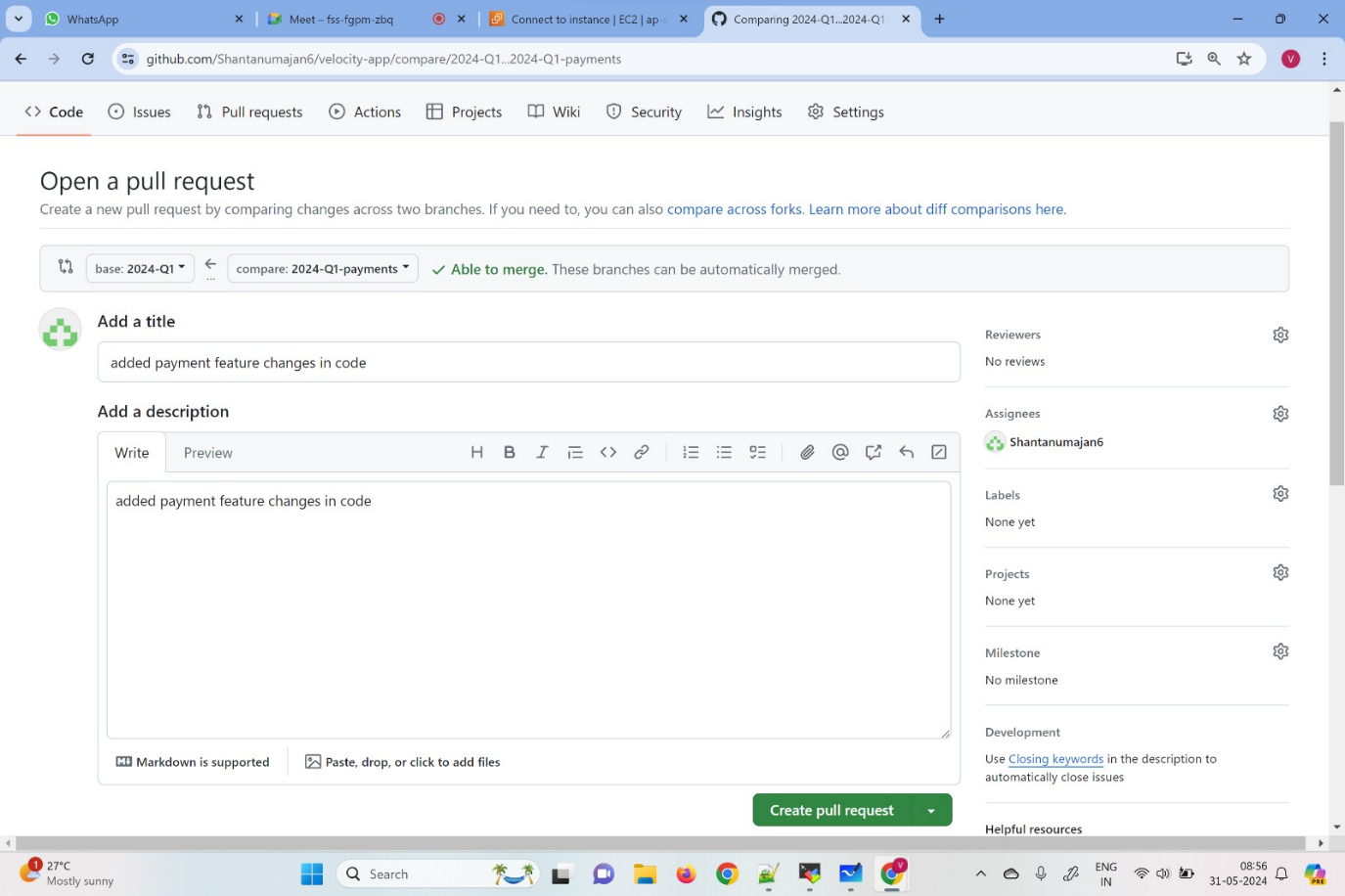
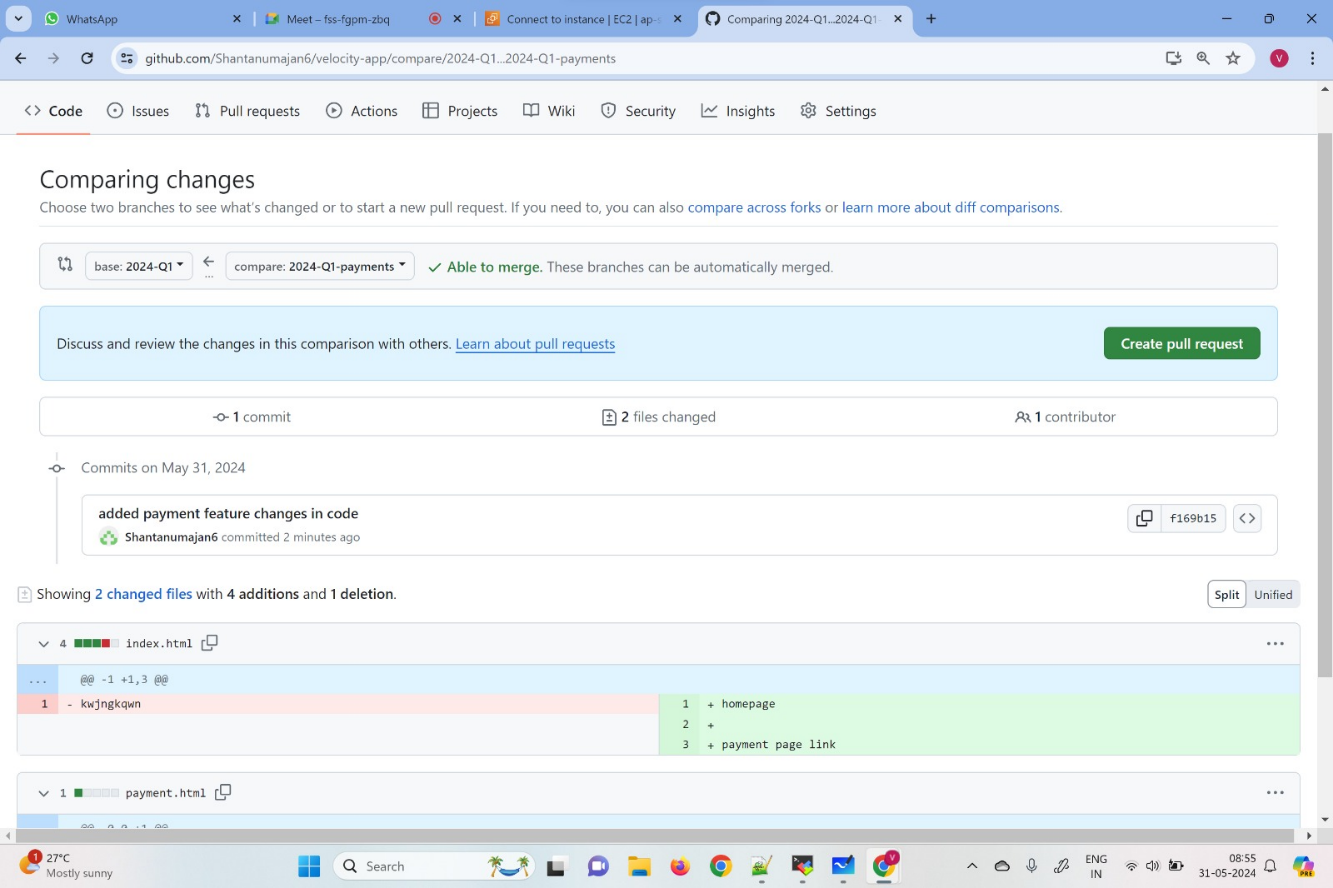
Then we create the new branch 2024-q1

And suppose we have another payment feature and we need to work then we need to clone the repo and create new branch 2024-q1-payments and make the changes what we want

Then as per the company standard we can’t merge the 2024-q1 branch and 2024-q1-payment branch .

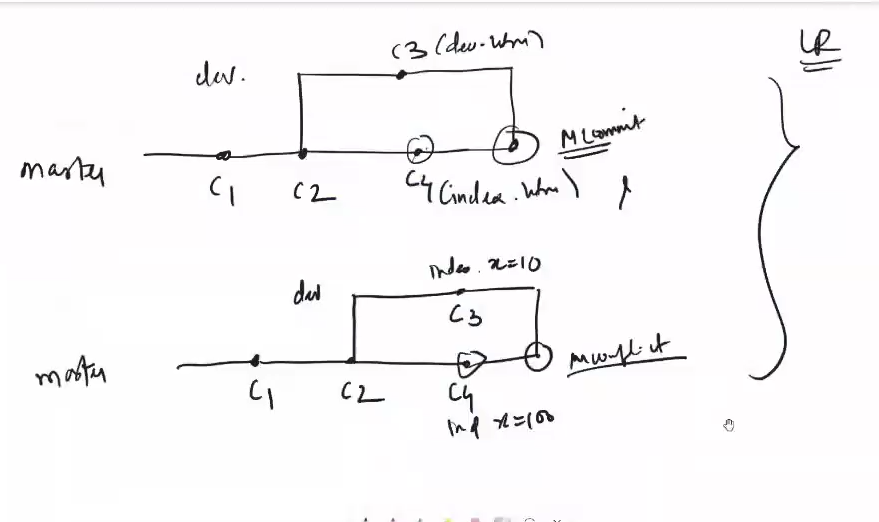
We need to raise the pull request to merge the changes. And we add the reviewers then they verify the full request and then accept

If we change the same index.html file from both branch and commit the changes and wants to merge then need to create pull request then there will be conflict so we need to resolve the conflict uding resolve conflict tab then merge

** **

**Merge commit---if changes are within diff files**

**Merge conflict—changes in same file**

****

**1 yum install git -y**

**2 clear**

**3 git config --global user.name shantanu**

**4 git config --global user.email shantanumahajan6@gmail.com**

**5 mkdir vel-app**

**6 cd vel-app/**

**7 git init**

**8 vi index.html**

**9 git add index.html**

**10 git commit -m "index file added"**

**11 vi dev.html**

**12 git add dev.html**

**13 git commit -m "addded dev file"**

**14 git log**

**15 git branch dev**

**16 git checkout dev**

**17 vi dev.html**

**18 git add dev.html**

**19 git commit -m "added dev change"**

**20 git log**

**21 git checkout master**

**22 vi index.html**

**23 git add index.html**

**24 git commit -m "added master change"**

**25 git log**

**26 git merge dev**

**27 git log**

**28 git checkout dev**

**29 git merge master**

**30 git log**

**31 cd ../**

**32 rm -rf vel-app/**

**33 clear**

**34 mkdir vel-app**

**35 cd vel-app/**

**36 git init**

**37 vi index.html**

**38 vi dev.html**

**39 git add \***

**40 git commit -m "Added 2 files"**

**41 git log**

**42 git branch dev**

**43 vi index.html**

**44 git add index.html**

**45 git commit -m "Added index file"**

**46 git checkout dev**

**47 vi index.html**

**48 git add index.html**

**49 git commit -m "Added x=100"**

**50 git log**

**51 git checkout master**

**52 git log**

**53 git commit --amend -m "x=10"**

**54 git log**

**55 git merge dev**

**56 cat index.html**

**57 git merge --abort**

**58 cat index.html**

**59 git merge dev**

**60 cat index.html**

**61 git merge --abort**

**62 cat index.html**

**63 cd ../**

**64 rm -rf vel-app/**

**65 mkdir velocity-app**

**66 cd velocity-app/**

**67 git init**

**68 vi index.html**

**69 git add index.html**

**70 git commit -m "Added index.html"**

**71 git remote add origin https://github.com/Shantanumajan6/velocity-app.git**

**72 git push origin master**

**73 cd ../**

**74 rm -rf velocity-app/**

**75 clear**

**76 git clone https://github.com/Shantanumajan6/velocity-app.git -b 2024-Q1**

**77 ls -ltr**

**78 cd velocity-app/**

**79 git branch**

**80 git branch 2024-Q1-payments**

**81 git branch**

**82 git checkout 2024-Q1-payments**

**83 git branch**

**84 vi payment.html**

**85 ls -ltr**

**86 vi index.html**

**87 git add \***

**88 git commit -m "added payment feature changes in code"**

**89 git log**

**90 git push origin 2024-Q1-payments**

**91 cd ../**

**92 rm -rf ve**

**93 rm -rf velocity-app/**

**94 clear**

**95 git clone https://github.com/Shantanumajan6/velocity-app.git**

**96 ls -ltr**

**97 cd velocity-app/**

**98 ls -ltr**

**99 cat i**

**100 cat index.html**

**101 clear**

**102 vi index.html**

**103 cat index.html**

**104 git checkout index.html---revert the changes and keep the original copy in LR**

**105 cat index.html**

**106 clear**

**107 ls -ltr**

**108 touch 1 2 3 4**

**109 ls -ltr**

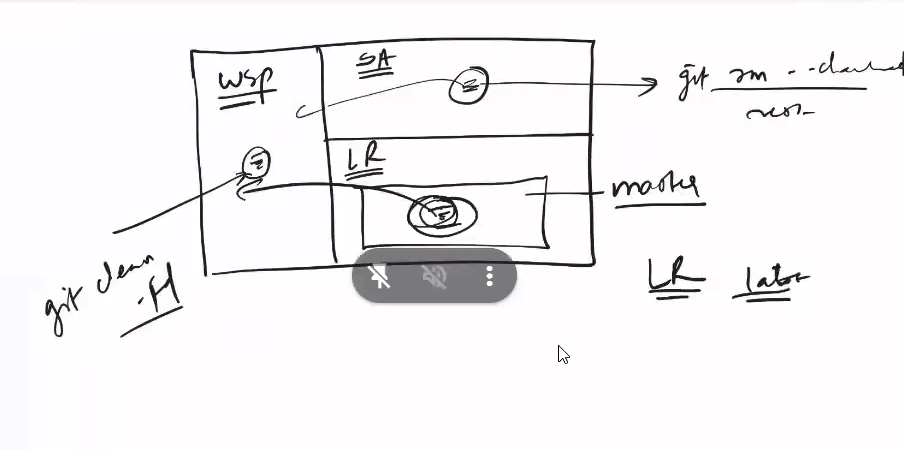
**110 git status**

**111 git clean -fd—delet the unwanted files which is in wsp**

**112 history**

**Fork: fork action creates a copy of the another account repo under our own github acc**

**Interview specific cmd:**

****

**git reset:- when we are in Local repository**

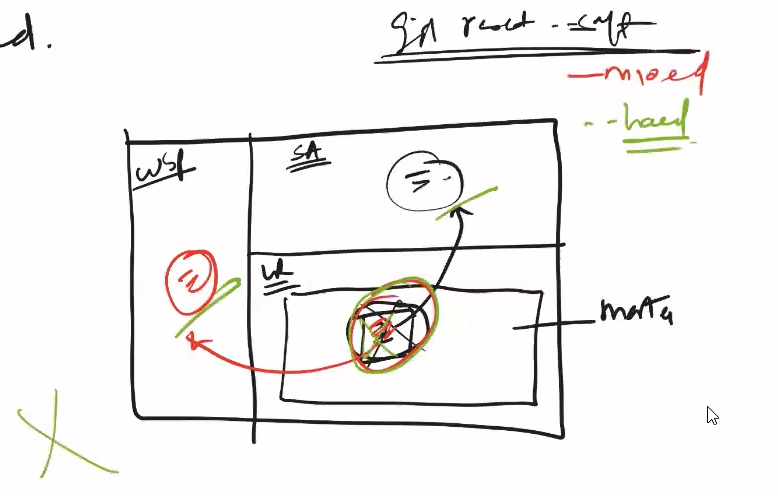
**git reset HEAD~1 -- to delete the latest commit**

**git reset HEAD~1 -- delete latest 2 commit**

**three types:- mixed (default), Hard, soft**

**soft—delete change from lR and go to SA**

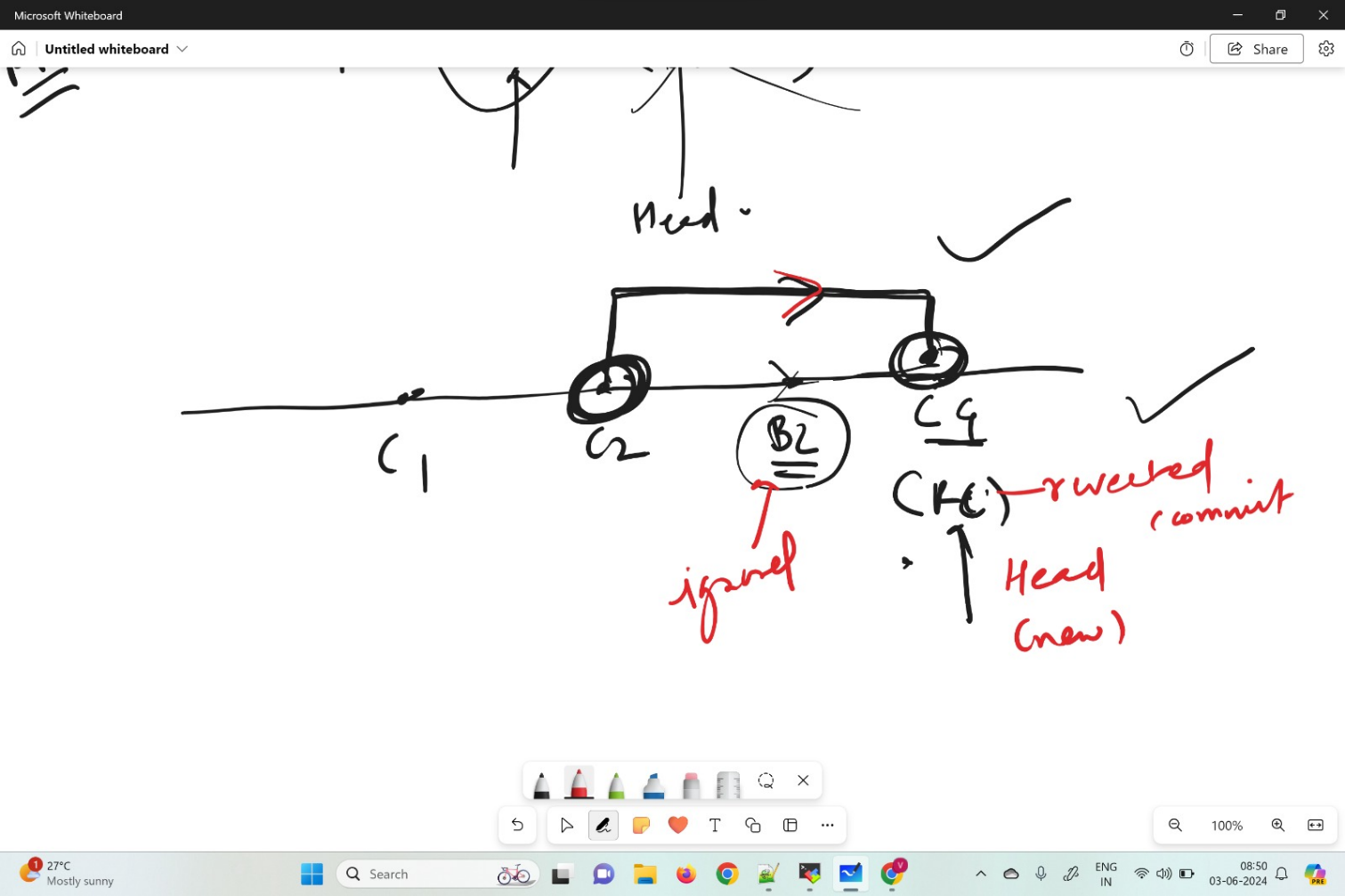
**mixed- delete change from a LR and come in WSP**



**hard- permently delete change**

**git reset HEAD~1 --soft**

**git revert:**

****

**git revert commitID**

**then add comment to the pop-up.**

**git tag tagname commitID -- lightweight**

**git tag tagname commitID -m "message to write" -- heavy**

**git tag -------------- can list**

**git show tagname**

**git tag -d tagname**

**git push origin master --tags**

**git log**

**git log --oneline**

**can have multiple origins:-**

**git remote add origin2 url**

**here origin2 is nothing but a name...can be anything but prefer origin**

