Module 1: Introduction to Devops and Core Concepts

=====================================================

Trainer: Sonal Mittal

===========================================

Session1:

==================

Discuss on SDLC and its phases

Discuss on Traditional Models of SDLC

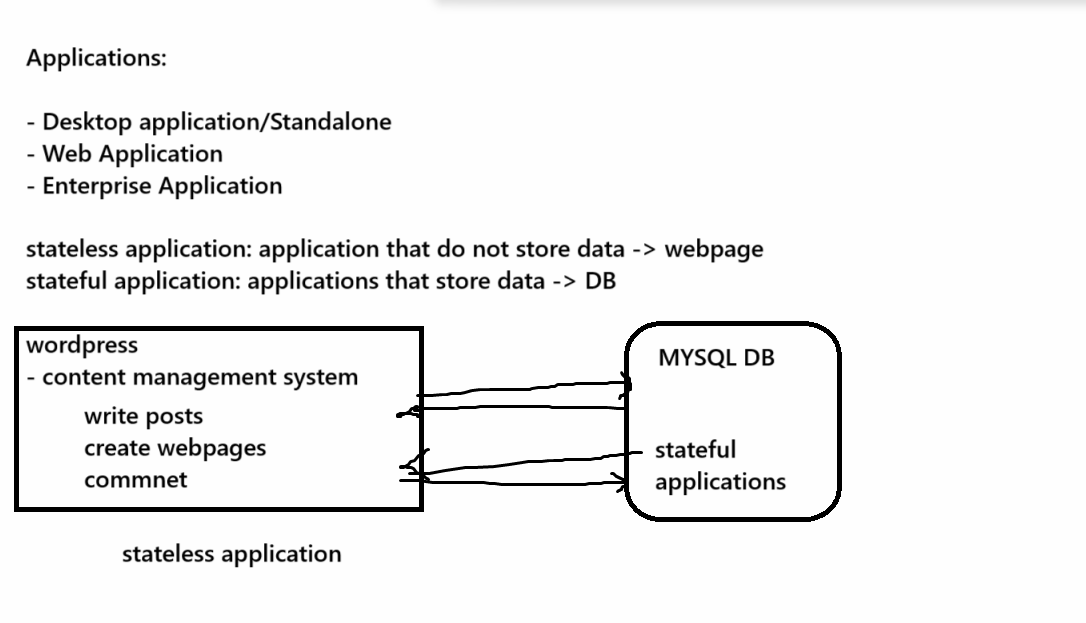
Introduction to DevOps and its workflows

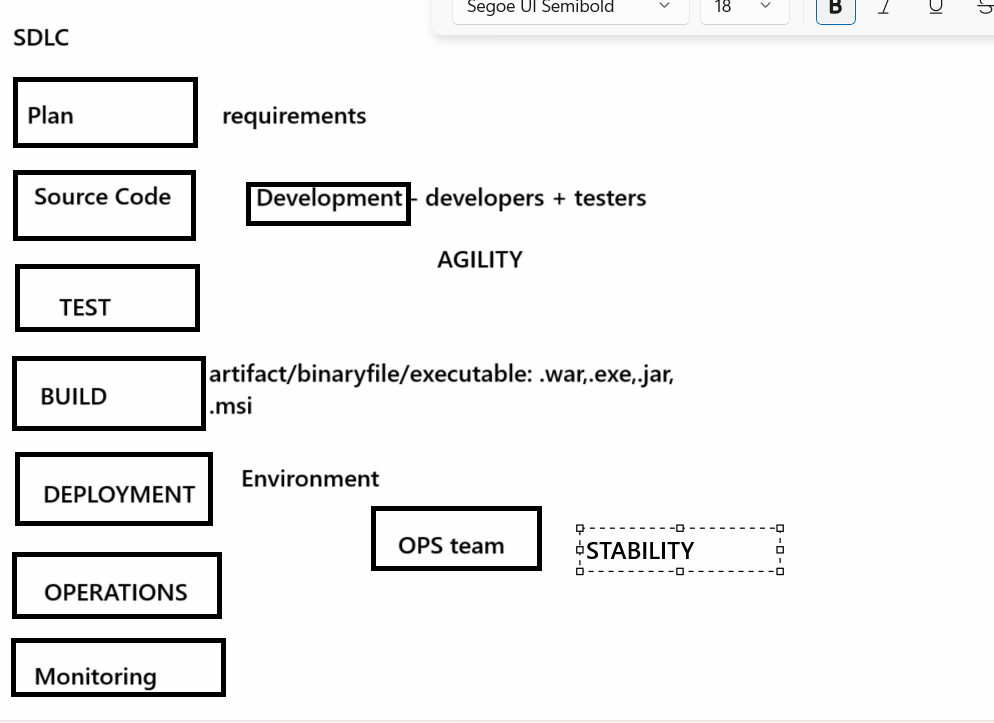
Discuss and Explain principles of DevOps

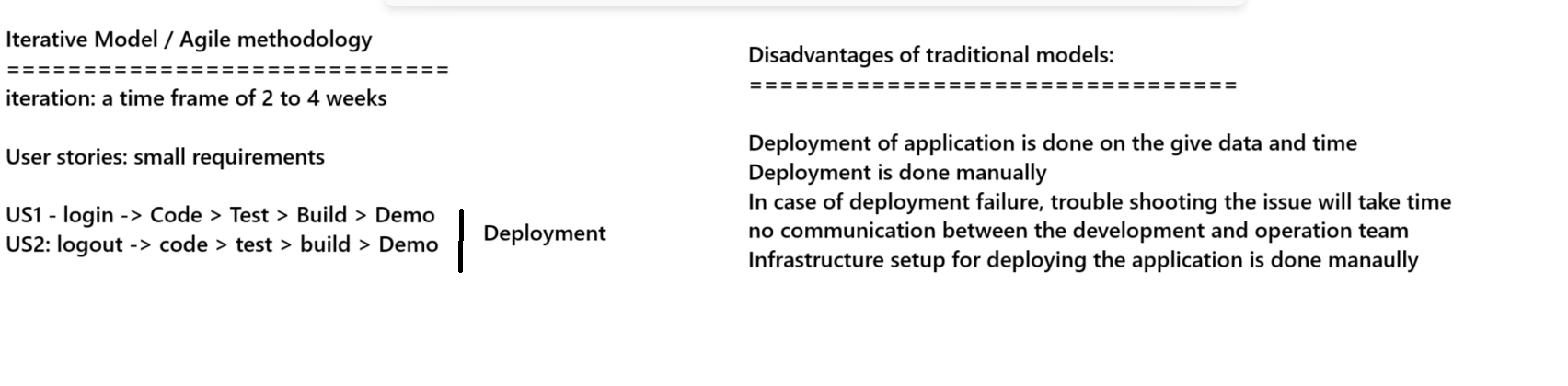
Explain Devops tools and Devops Pipeline

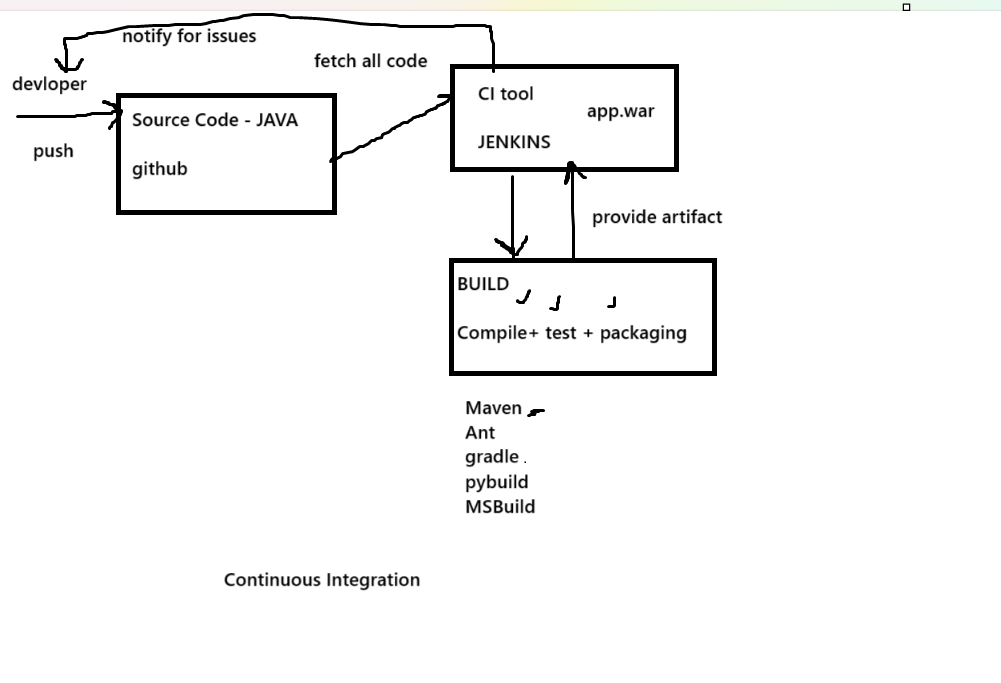
Introduction to Version Control and Version Control systems

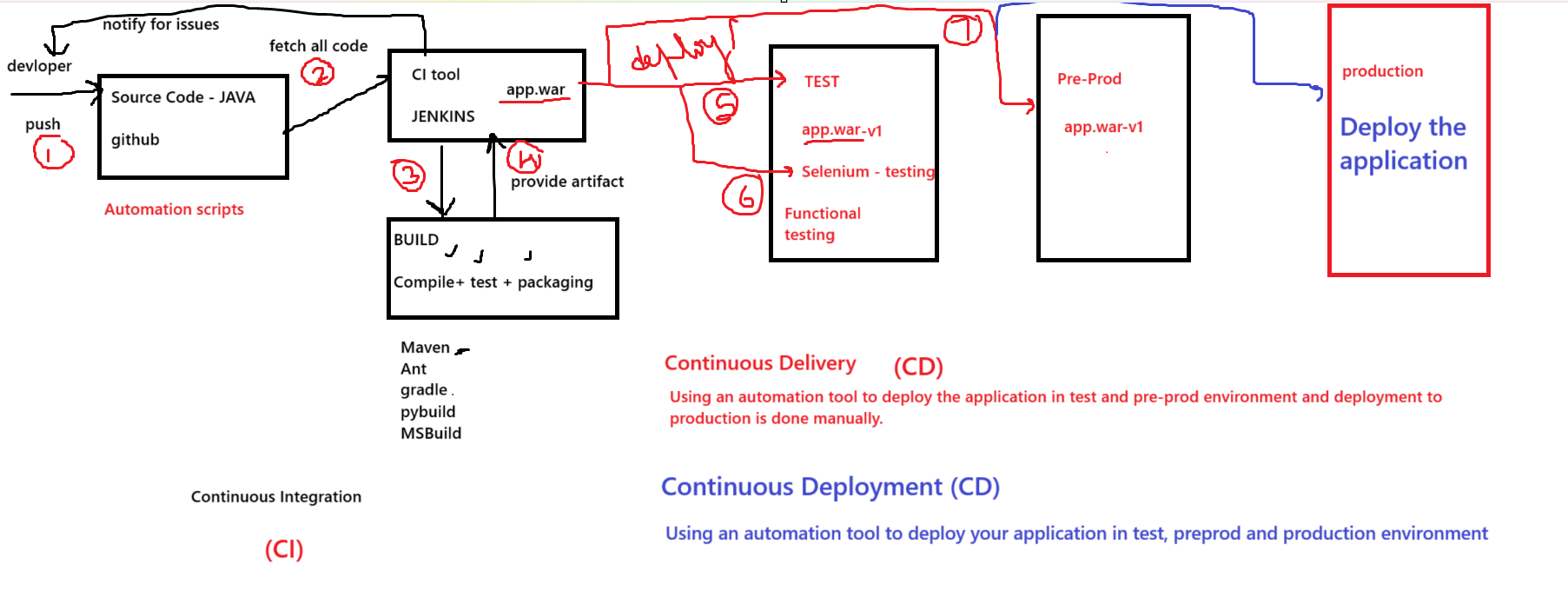
Discuss Git workflows

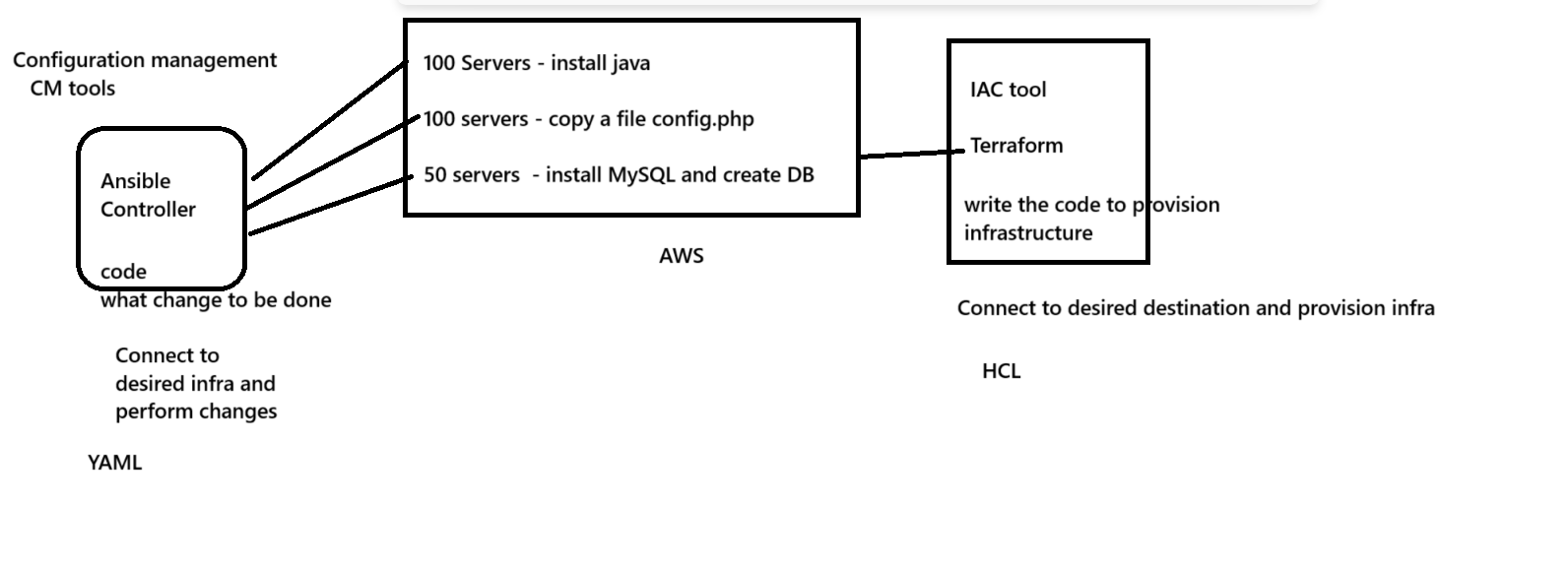




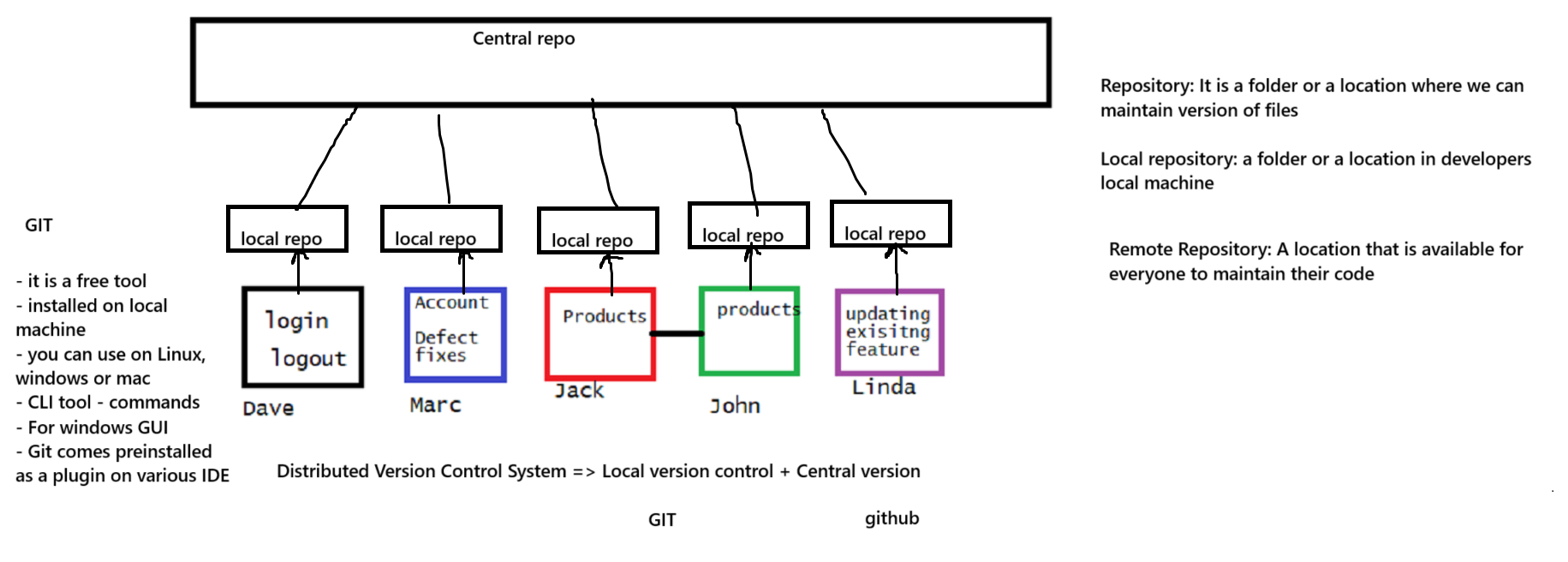


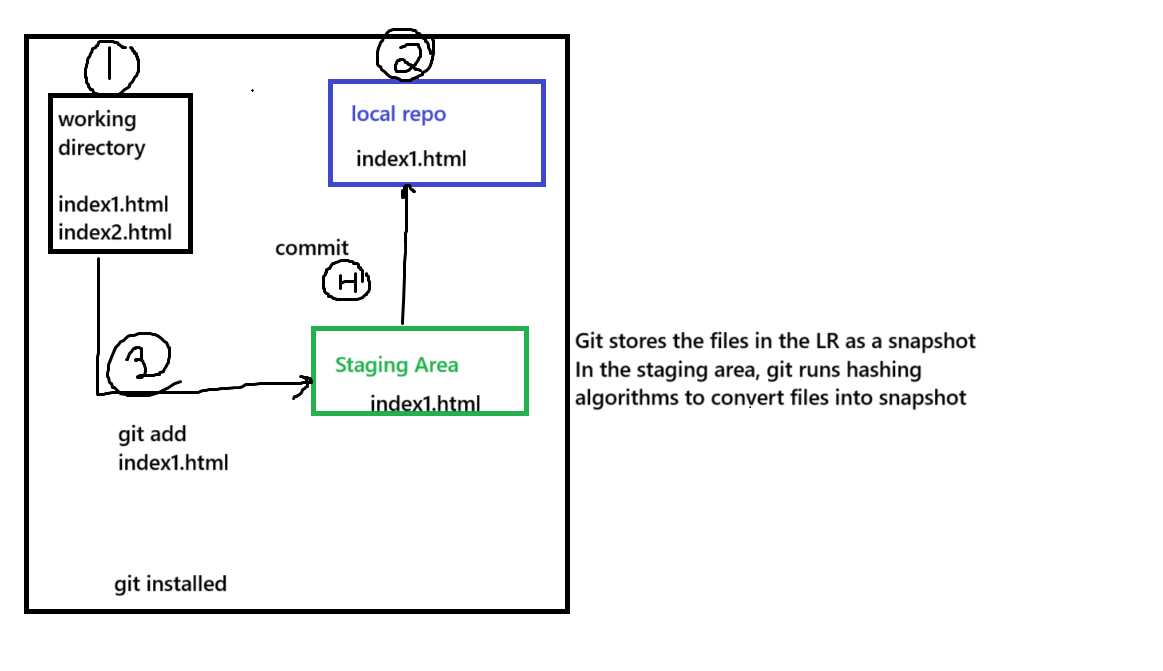






Continuous Version Control





Check version of git:

# sudo su -

# git --version

**Demo 1: Create a working directory and initiate a local repository**

**# mkdir myproject && cd myproject**

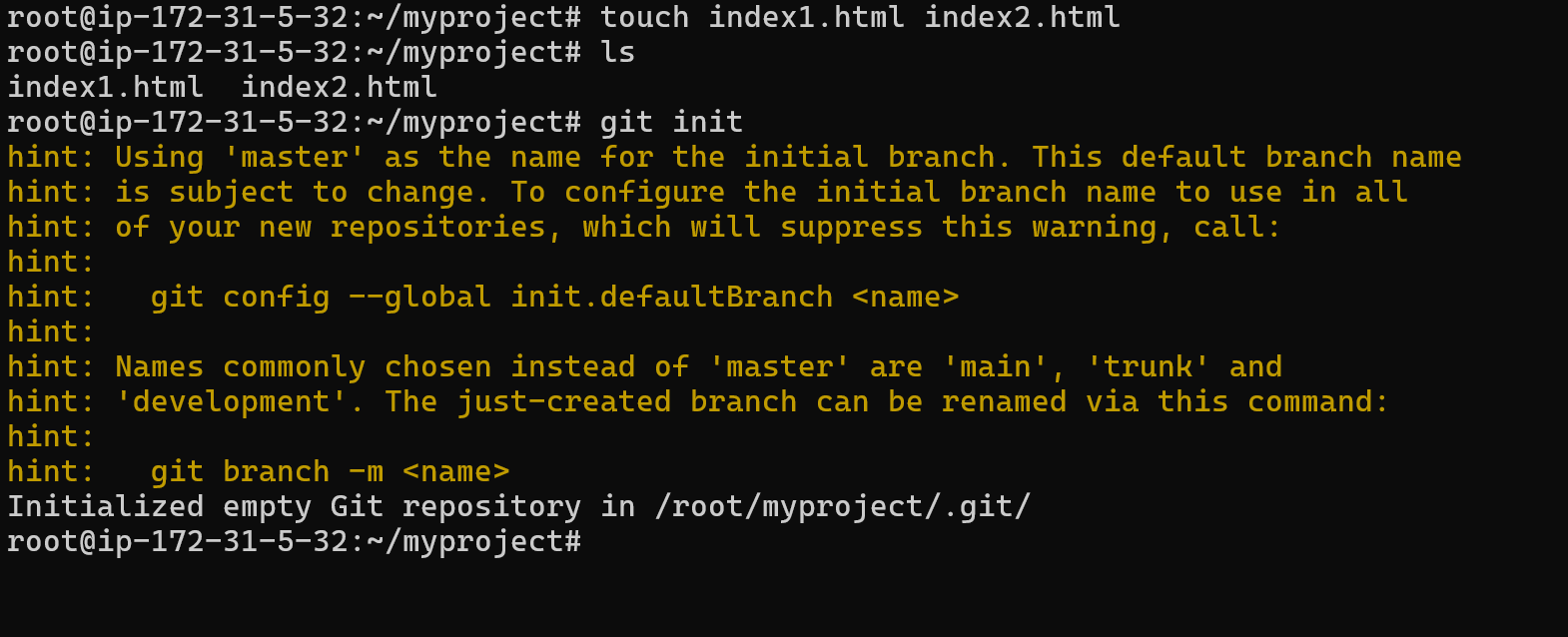
**# touch index1.html index2.html**

**Create local repo**

**# git init**

**Initialized empty Git repository in /root/myproject/.git/**

**.git is name of local repo**

****

**Demo 2: Git configurations: It is mandatory to setup configurations**

**=====================================**

**To setup author name and email id of user who is committing into local repo**

**# git config --global user.name sonal04**

**# git config --global user.email** [**sonal@gmail.com**](mailto:sonal@gmail.com)

**# git config --list**

**# cat ~/.gitconfig**

**There are 3 types of git config that we can do on git**

1. **Local :**

**Every repository on the VM can have its own configuration**

**#  git config --local user.name admin**

**cat .git/config**

1. **Global**

**Commits to any repository created by current user logged in the VM these config will be applicable**

**# git config --global user.name globaluser**

**# cat ~/.gitconfig**

1. **System**

**Same configuration applicable to every repository created by any user on the VM**

**Demo 3:  Add a file to Local repo**

**====================================**

**Check the status of wroking directory**

**# git status**

**Add the files to staging area**

**# git add index1.html**

**# git status**

**Commit the file to Local repo**

**# git commit -m "added file index1.html"**

**See the git log and configurations**

**# git log**

**=========================================**

**Day 2: 09 March 2025**

**=========================================**

**> Modify a tracked file and commit it to Local repo**

**> Restore the file**

**> deletion of file**

**>revert the commit**

**> Reset operations**

**=========================================**

**# sudo su -**

**# cd myproject**

**# git status**

**If any files are untracked**

**# git add .**

**#  git commit -m "added file index2.html"**

**# git log --oneline**

**Check the files that are tracked by git**

**# git ls-files**

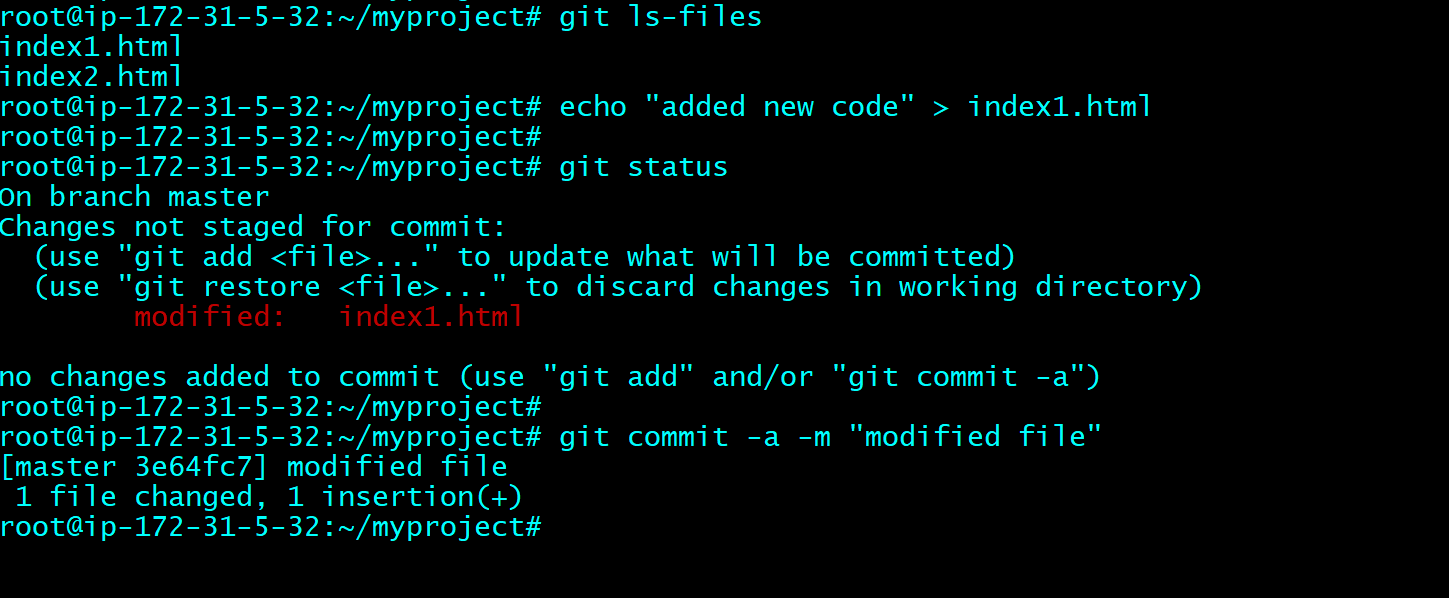
**Demo 1: Modify an existing tracked file and commit to the repository:**

**# echo "added new code" > index1.html**

**# git status**

**Directly add the changes and commit to LR**

**#  git commit -a -m "modified file"**

****

**Demo 2: See the difference between 2 version of the file index1.html**

Make changes to an existing file that is tracked by git

Add the modifications to the local repo

**# echo "Added new content in the file" >> index1.html**

**# git status**

**# git diff index1.html**

**Restore workflow:**

**==========================**

**Discard all the modified lines from a file in working directory**

**The changes are removed permanently, you cannot get them back.**

**# git restore index1.html**

**# git status**

**Check the content of the file to see that changes are not available now. It is a linux command**

**# cat index1.html**

**For you information, another command to discard changes from WD:**

**# git checkout -- index1.html**

**Demo 3:**

**================================**

**Stage the modifications and see the difference**

Make changes to an existing file that is tracked by git

Add the modifications to the local repo

**# echo "Added new content in the file" >> index1.html**

**# git status**

**Stage the modifications**

**# git add .**

**# git diff --staged index1.html**

**Move the changes from staging area to unstaged area or working directory**

# git restore --staged index1.html

# git status

=====================================

# git status

#  git commit -a -m "done modification"

Demo 4: Remove the file from working directory and local repo

# git rm index1.html

# git status

File is not there in local repo or Working directory

# ls

# git ls-files

Git will track your deleted files also

# git commit -m "deleted the file"

# git log --oneline

===============================

Revert in GIT

================================

1. This command is applied on a single commit id
2. What ever change we have done on a commit,we can always revert it to its previous version
3. Git show command will tell us what we have added/deleted from Local repo
4. Revert command will generate a new commit ID
5. Because there is a new commit id, we have to give commit message
6. When revert command is executed → an editor will open on the terminal → Enter a message on the editor -> save the file on editor → a new revert commit id will be created

**# git status**

**You working tree should be clean**

**# git ls-files**

**Delete a file that is in local repo and working directory**

**Git command to delete a file from LR and WD**

**# git rm index1.html**

**# git status**

**# git commit -m "deleted file"**

**# git log --oneline**

**# git revert <commitID>**

**Add message in the nano editor on first line and save the file**

**Save on nano editor**

**Press CTL x**

**Press y**

**Press enter key**

**A new commit will be generated for the revert operation**

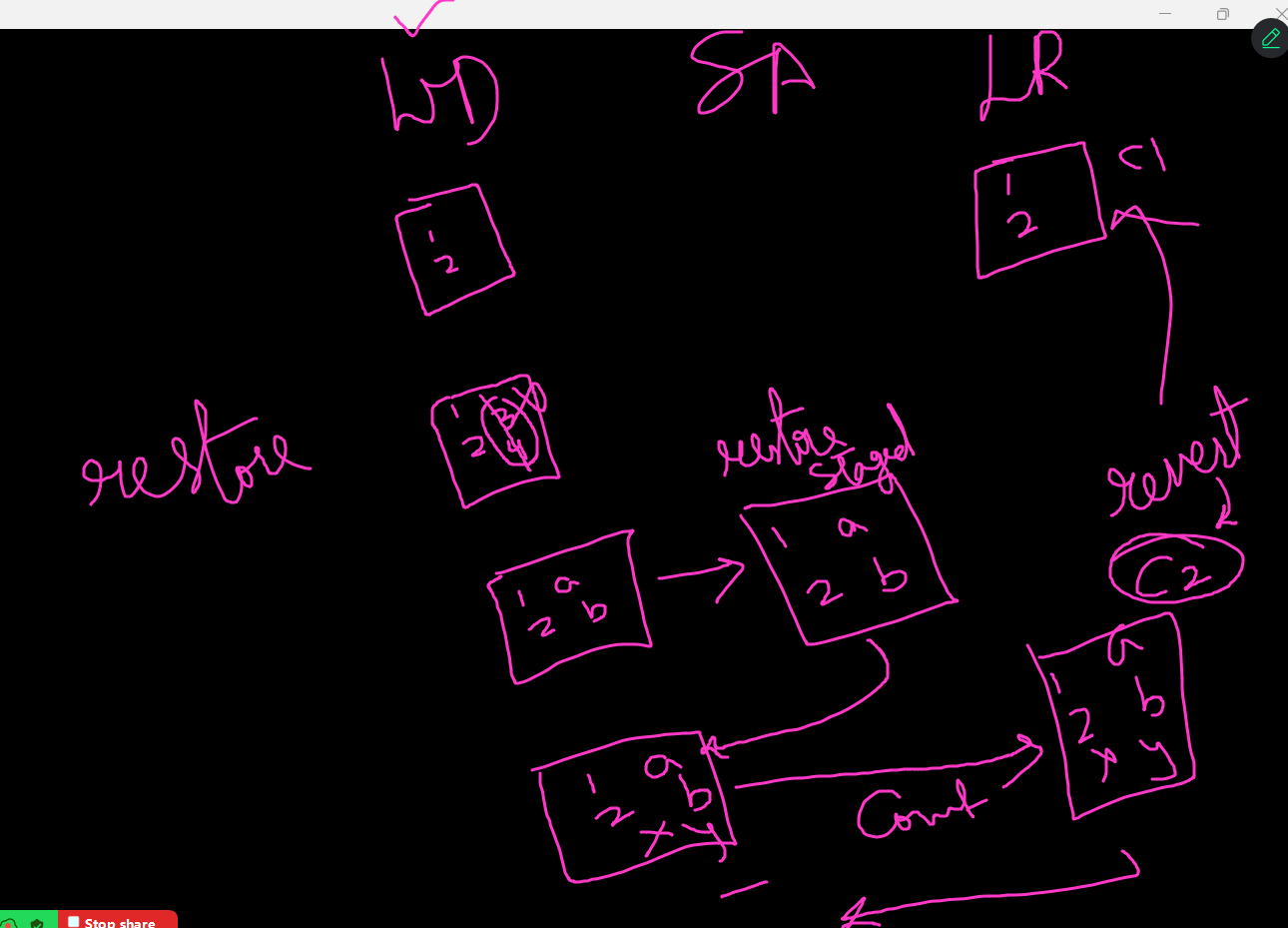
**# git log --oneline**

**File is back**

**# ls**

**# git ls-files**

**============================================**



RESET Command

======================

1. Reset command can reset the HEAD to a different commit id
2. This command will delete commits
3. This command will not generate any new commit id

**We always want a clean and crisp commit history**

**Git says, you can RESET your commit history**

**that means:**

**> we can Change the HEAD of the log**

**> we can delete commit ids**

**> we can move the changes from LR to Staging area**

**> we can move the changes from LR to Working area**

**> we can also permanently delete commits and changes in**

**LR and working directory**

**the command is : git reset <commit id>**

**What is it that we have to reset,**

**> we want to reset the HEAD to the given <commit id>**

**#  git reset d573778**

**Now this d573778 becomes the HEAD**

**So what about the commits before the given the <commit id> --> they will get deleted**

**So what about the changes we did as part of those commits?**

**Soft:**

**=======================**

**# echo "add content" >> index1.html**

**# git add .**

**# git commit -m "done modification1"**

**# echo "add content again " >> index1.html**

**# git add .**

**# git commit -m "done modification2"**

**# echo "add content again 123 " >> index1.html**

**# git add .**

**# git commit -m "done modification3"**

**# git log --oneline**

**# git reset --soft <commitID>**

**--soft:**

**Observation:**

**> In the git log, the commit history is reset -> all the commits which are above the given <commitID> get deleted**

**> Now the HEAD is the  <Commitid> given in the command**

**> All the changes of the deleted commits will be back to the staging area.**

**# git commit -m "all modifications together"**

MIXED:

=================

# git log --oneline

Choose a commit id

**# git reset --mixed <commit id>**

**--mixed**

**Observation:**

**> In the git log, the commit history is reset -> all the commits which are above the given <commitID> get deleted**

**> Now the HEAD is the  <Commitid> given in the command**

**> All the changes of the deleted commits will be back to working directory**

**# git add .**

**# git commit -m "all modifications together"**

**# git status**

====================

HARD

**--hard**

**=============================**

**Observation:**

**> In the git log, the commit history is reset -> all the commits which are above the given <commitID> get deleted**

**> Now the HEAD is the  <Commitid> given in the command**

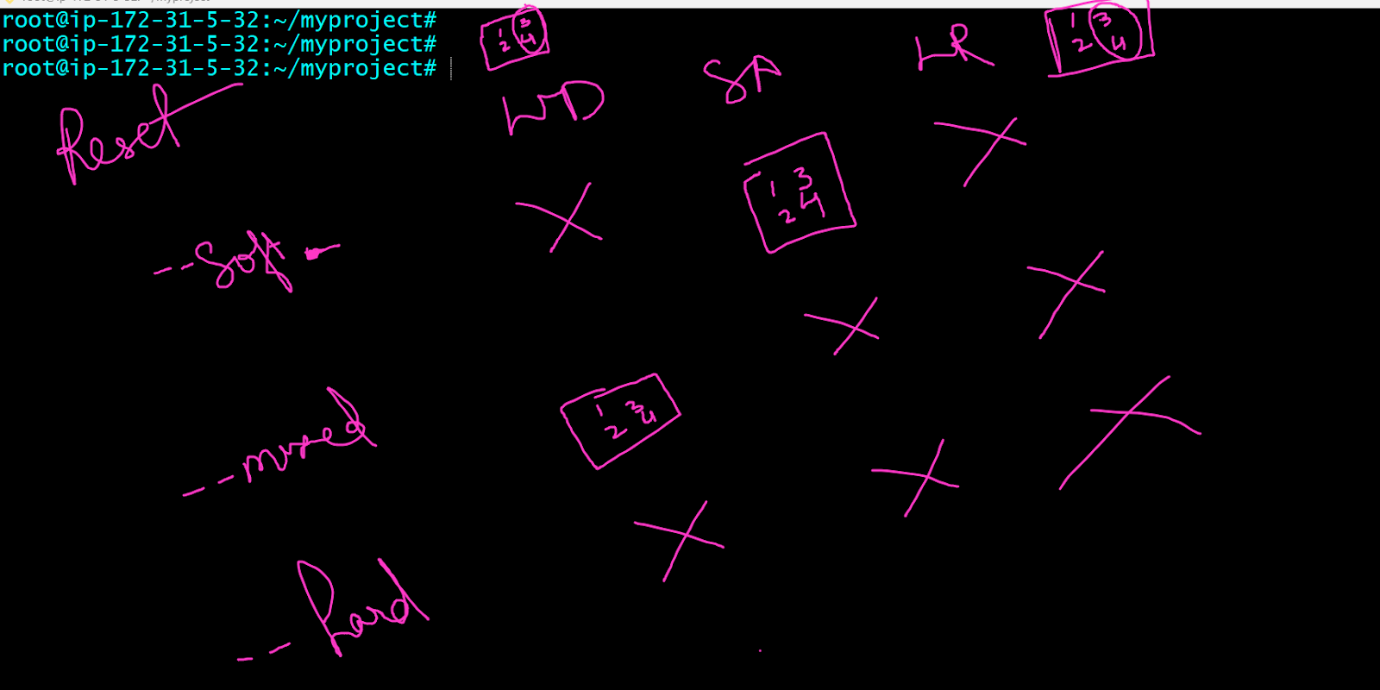
**> All the changes of the deleted commits will also be permanently deleted**

**# git log --oneline**

**Select the commit id which is at the bottom of your log.**

**# git reset --hard <commit id>**

**Example : git reset --hard 04f432e**



Day 3: 15 March 2025

=======================

> .gitignore file

> stash command

> branching and merging

Connect to the EC2 server:

# sudo su -

Search for your project directory

# ls

Go inside the project directory

# cd myproject

# git status

You should have clear working tree

Like this :

On branch master

nothing to commit, working tree clean

If any files are shown as untracked or to be committed then execute below commands:

# git add .

# git commit -m "done"

===============================

If you don't find your project, to proceed further

# mkdir mydemo

# cd mydemo

# git init

=================================

Demo 1: .gitignore concept

In git we can choose which files to be tracked and which files to be ignored.

Tracking of the file can be done by git add. → gitcommit -m

Ignoring of the files can done using .gitignore concept

This concept is applicable to untracked files only

This concept is not applicable to staged or committed files.

Execute below steps:

===============================

# touch file.log file.exe file.xml

# vim .gitignore

\*.tfstate

\*.xml

\*.log

\*.class

\*.exe

\*.jar

\*.war

Save the file (press esc key -> press :wq! -> press enter key)

# git status

Git will ignore files that we created

Git will track .gitignore file

#  git add .

#  git commit -m "added .gitingnore"

Demo2: Stashing in GIT

================================

We may have files in which I have made some changes but they are still in progress and are not ready to be staged or committed.

Suddenly I have a requirement as per which I have to work on another feature file right away and stop working on the current file

Since the changes in current file are incomplete, i do not wish to commit them in LR right now

However if I keep these incomplete changes in my working directory, there is a chance that I may execute (git add . or git commit -a -m  )command and all the incomplete changes will get staged and committed

So to handle this situation, git provides a concept of stash, where incomplete changes of a file or set of files can be moved to a temporary location(.git/refs/stash)

The changes will no longer be there in the working directory, so there is no chance of committing incomplete changes now.

In the future, we can always bring up the stashed changes to the working directory after which we can complete the work and commit them.

Execute below use cases:

Use case 1: One file with incomplete changes

======================================

#  git status

On branch master

nothing to commit, working tree clean

Take the file that has been committed to local repo

# vim index1.html

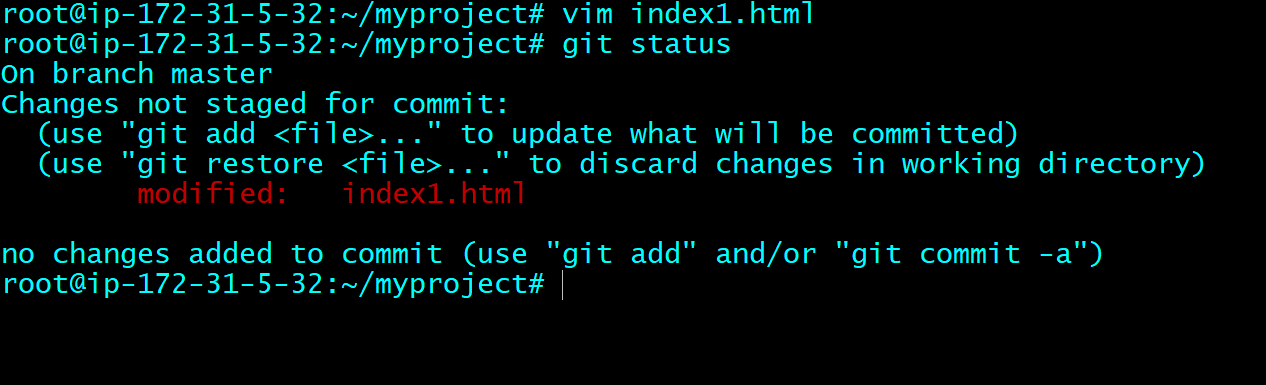
Add some content in the file

Save the file.

OR use echo command to add some content

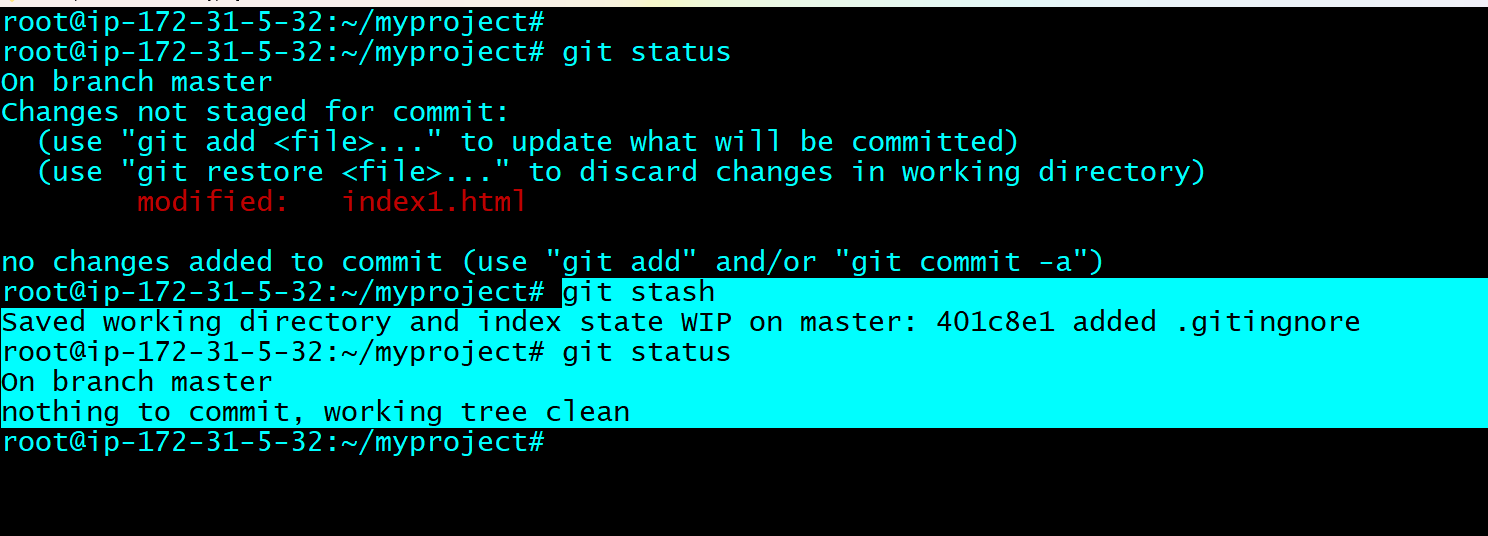
# echo "new content" >> index1.html

# git status



# git stash

 This command will take all modifications from working directory and stash them



To check your stash

# git stash list

To see what is stored in stash

# git show stash@{0}

To see the stash in the .git folder

# cat .git/refs/stash

Get back the changes from stash to working directory

# git stash pop stash@{0}

Use case 2: Multiple files with modifications, stash only a few changes.

Create some new files and commit to LR

# vim index2.html index3.html

# git add .

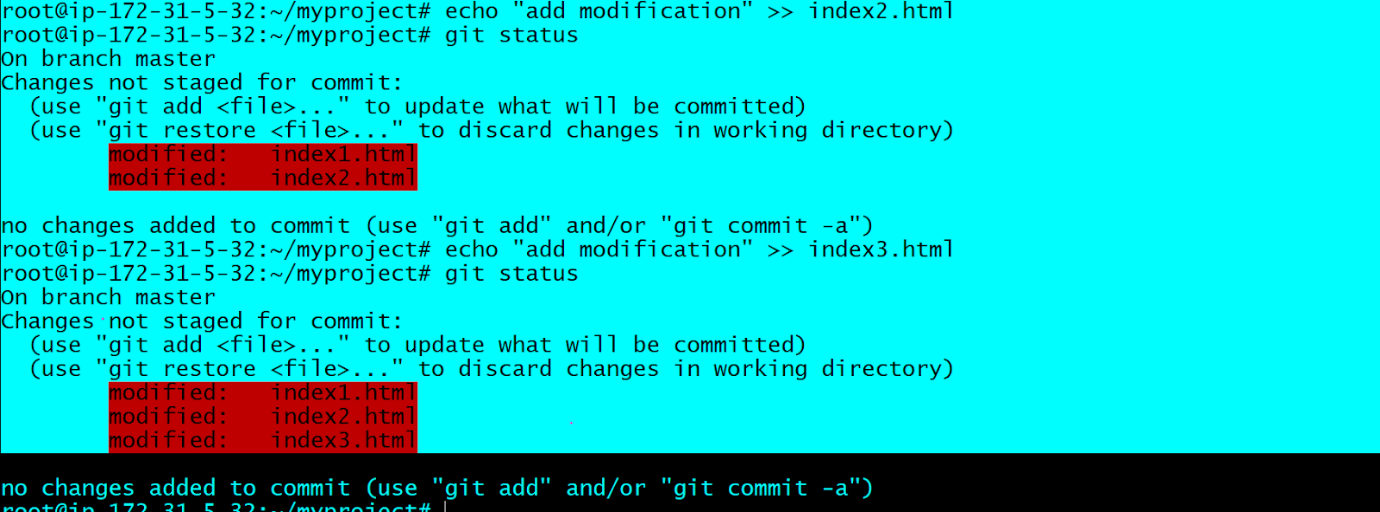
# git commit -m "add files"

Make modification to index2. and index3 file

# echo "add modification" >> index2.html

# echo "add modification" >> index3.html

# git status



Now stash changes of index1 and index 3

# git stash -p

Give y to stash the changes of index1.html → press enter key

Give n to not stash the changes of index2.html → press enter key

Give y to stash the changes of index3.html → press enter key

# git status

You will see index2.html file changes are in working directory

# git stash list

# git show stash@{0}

You will see index1.html file and index3 file changes are in stash

Bring back the changes

# git stash pop stash@{0}

In case we have to stash our changes in another directory and not in .git/refs/stash then we can use the command option:

# git stash push -- /path/to/folder

===================================

Demo 3: Branching and Merging

===================================

Before we begin this use case, let us setup or working directory and local repo

# git status

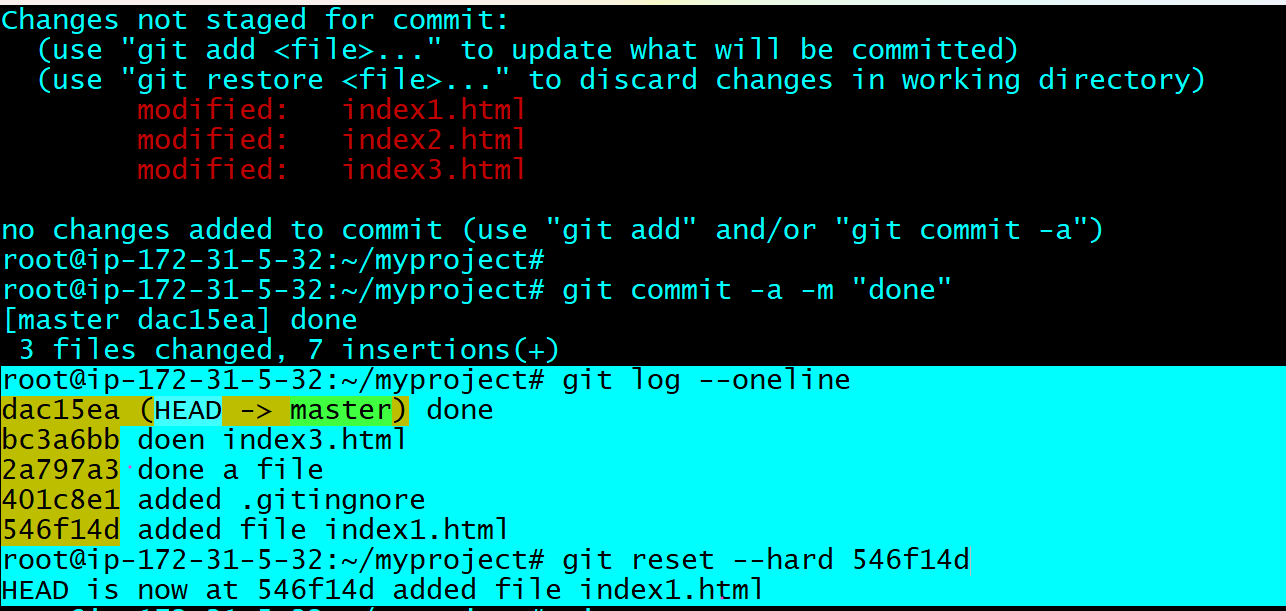
# git commit -a -m "done"

We will delete all the commits and all the changes

We will keep only 1 last commit id

Give your last commit id in this command

#  git reset --hard <commitid>



# git status

Any files are there → just commit them

# git add .

#  git commit -m "done"

Execute below commands:

========================

To see the branches we have on local repo

# git branch

Create a branch - method1

# git branch feature1

# git branch

To work on a branch, we have to switch to the branch

# git checkout feature1

# git branch

We will do new commits on feature1 branch

# touch file\_feature

# git add .

#  git commit -m "done on feature branch"

# git log --oneline

Note: this file and commit is not there on master branch

Now merge 2 branches:

=======================

Always checkout to the branch where we have to merge the changes

# git checkout master

# git merge feature1 master

All the changes form feature1 branch will be merged to master branch

We can delete the branch if required

# git branch -d feature1

Create a branch - Method2

============================

Create a new branch and also switch to the branch

#  git checkout -b dev

Create a file

# touch file\_feature

# git add .

#  git commit -m "done on feature branch"

# git log --oneline

Note: this file and commit is not there on master branch

Now merge 2 branches:

=======================

Always checkout to the branch where we have to merge the changes

# git checkout master

# git merge dev master

All the changes from dev branch will be merged to master branch

We can rename the branch if required

# git branch -m dev feature1

Here -m = modify

# git branch

======================================

Day 4: 16-March-2025

======================================

Agenda:

> cherry-pick

> rebase

> conflicts and resolve merge conflicts

> tags

> remote repository

==================================

Connect to your EC2 server and execute below commands

# sudo su -

# ls

Whatever is your working directory, go inside that directory

# cd myproject

Query for branch and then delete all branches except for master

# git branch

# git branch -d <YOUR\_branchName>

On the master branch, delete all the commits except for the last one

# git log --oneline

Copy the last commit id

# git reset --hard <last\_commit\_id>

Demo 1: Cherry-pick

1. Create a branch and switch to the branch

#  git checkout -b dev

# git branch

You will be switched to dev branch

1. Now we will create some new commits on the dev branch

# touch file\_dev

# git add .

# git commit -m "done file\_dev"

# touch file\_dev1

# git add .

# git commit -m "done file\_dev1"

# touch file\_dev2

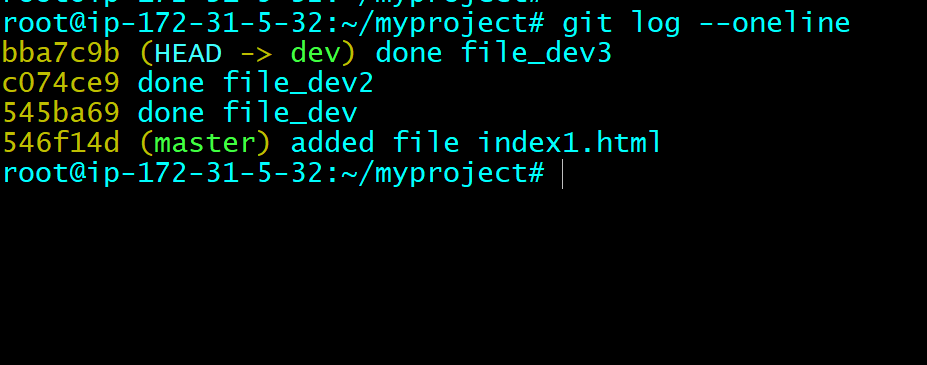
# git add .

# git commit -m "done file\_dev2"

Now you will see we have have new commits on the dev branch, these commits are not there on master branch

# git log --oneline

Copy the first commit id, that you want to merge to master.



1. We will now switch to master branch for merging the files and commits from dev branch

# git checkout master

# git branch

Give the commit id that was copied from dev branch git log --oneline command.

# git cherry-pick <commit\_id\_dev\_branch>

You will see the dev file is now present on master branch

# ls

Delete dev branch even though some of the changes on that branch have not been merged to master

In this case we are forcibly deleting the unmerged branch

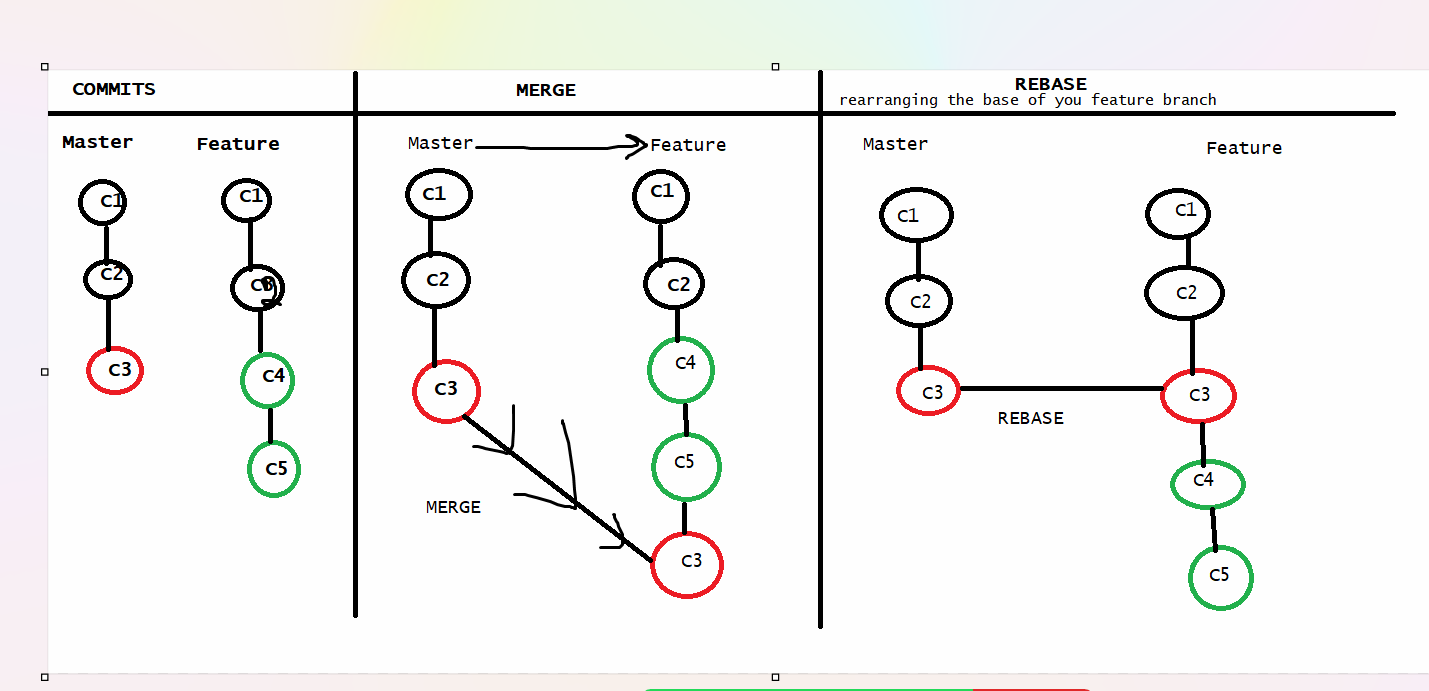
# git branch -D dev

# git branch

===================================

      Demo: Rebase merge strategy

===================================



For us the perform this demo, we need single commit on the master branch

Execute the log command and copy the last commit id

# git log --oneline

Copy the last commit id

# git reset --hard <last\_commit\_id>

You should now have only 1 commit on master branch

# git log --oneline

Now for rebase use cases, lets create a new branch. Do not switch to the branch

# git branch feature

# git branch

Creeate a new commit on master branch

# touch login

# git add .

# git commit -m "done on master"

# git log --oneline

You will have 2 commits now

Go to feature branch and create 2 new omits

# git checkout feature

# touch file1

# git add .

# git commit -m "done on feature"

# touch file2

# git add .

# git commit -m "done on feature"

Make sure you are checked out on the feature branch

# git branch

Now we will execute rebase command, that will rewind commit history by first placing all commits of base branch i.e. master branch and then new commits of feature branch

Rebase command is always executed on feature branches and not on master branch

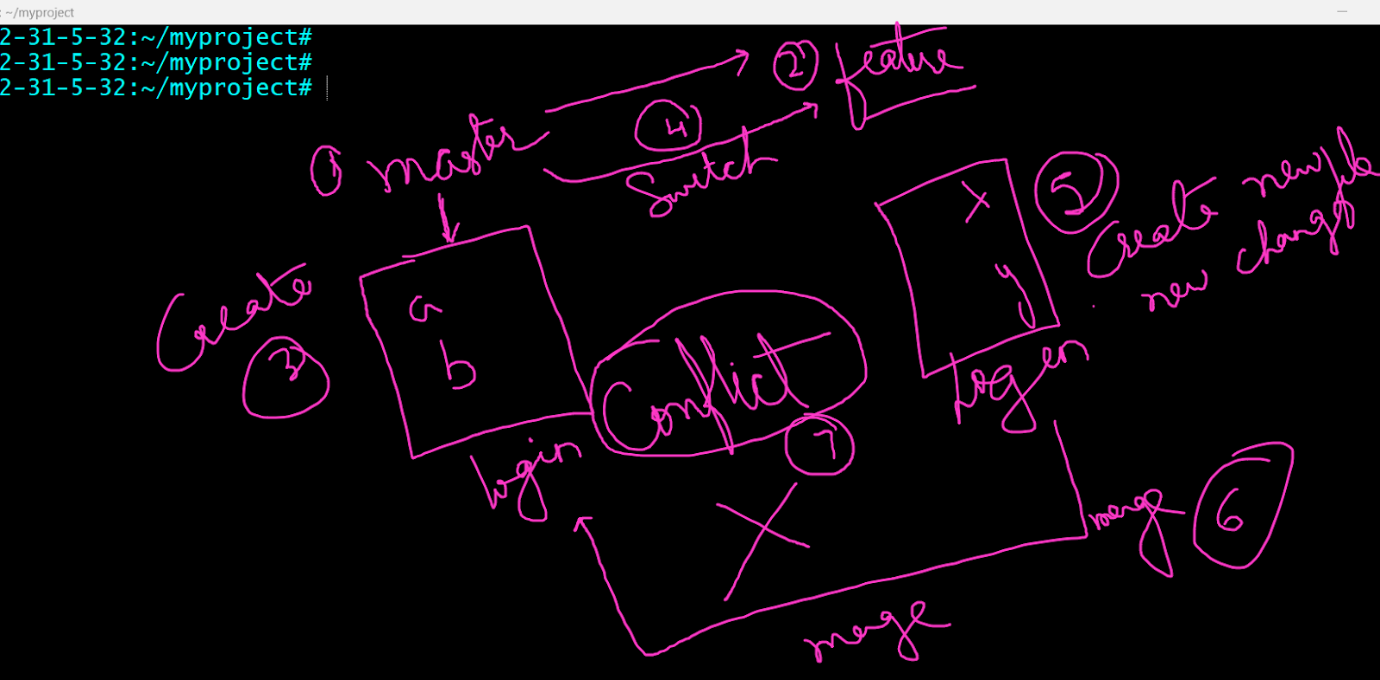
#  git rebase master

# git log --oneline

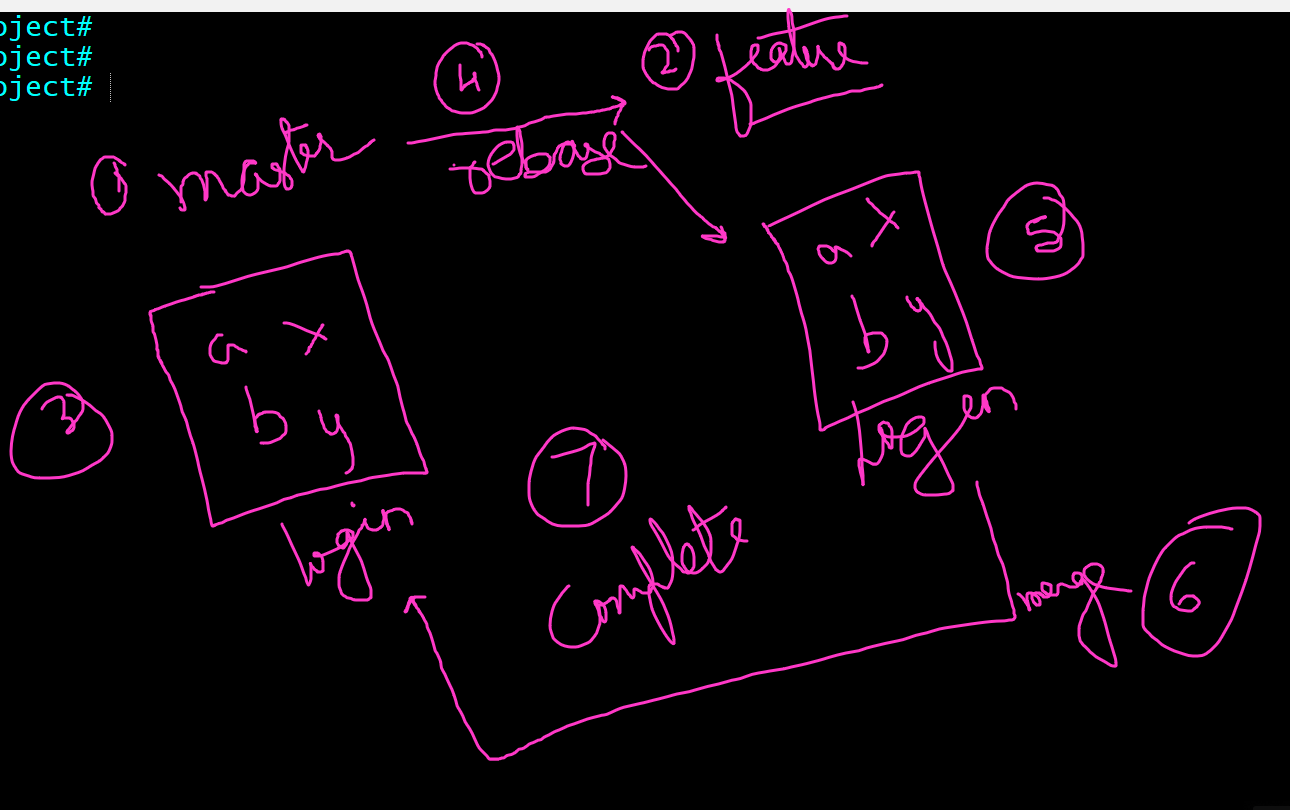
===================================

Demo: Merge conflict and resolve merge conflicts

When we get conflicts



To avoid conflicts



Demo:

Delete any branches you have, just master branch should be there

# git branch -D <branch\_name>

Delete the commits on master and you should have a single commit id

# git log --oneline

Copy the last commit id

# git reset --hard <last\_commit\_id>

Now you will have single branch - master

1. Create a new branch

# git branch feature

1. On master branch create a file with login and commit

# touch login

Add some content

# echo "changes on master" > login

# git add .

# git commit -m "changes on master"

1. Switch to branch feature

# git checkout feature

1. On feature create a file with same name as login but differen content

# touch login

Add some content

# echo "changes on feature" > login

# git add .

# git commit -m "changes on feature"

1. Now switch to master branch to do merge

         # git checkout master

# git merge feature master

But here merge will not complete, we will get CONFLICT

1. We have to resolve the conflict

Open the login file

# vim login

Remove lines with special characters line === HEAD  and  >>>> feature

# git status

You will see unmerged paths on the file

1. Commit the merged changes

# git commit -a -m  "resolve conflict"

==================================

GITHUB - Day 5-  22 March 2025

=================================

GITHUB account

Remote repository

Push in to remote repo using Personal Access token

Push in to remote repo using SSH

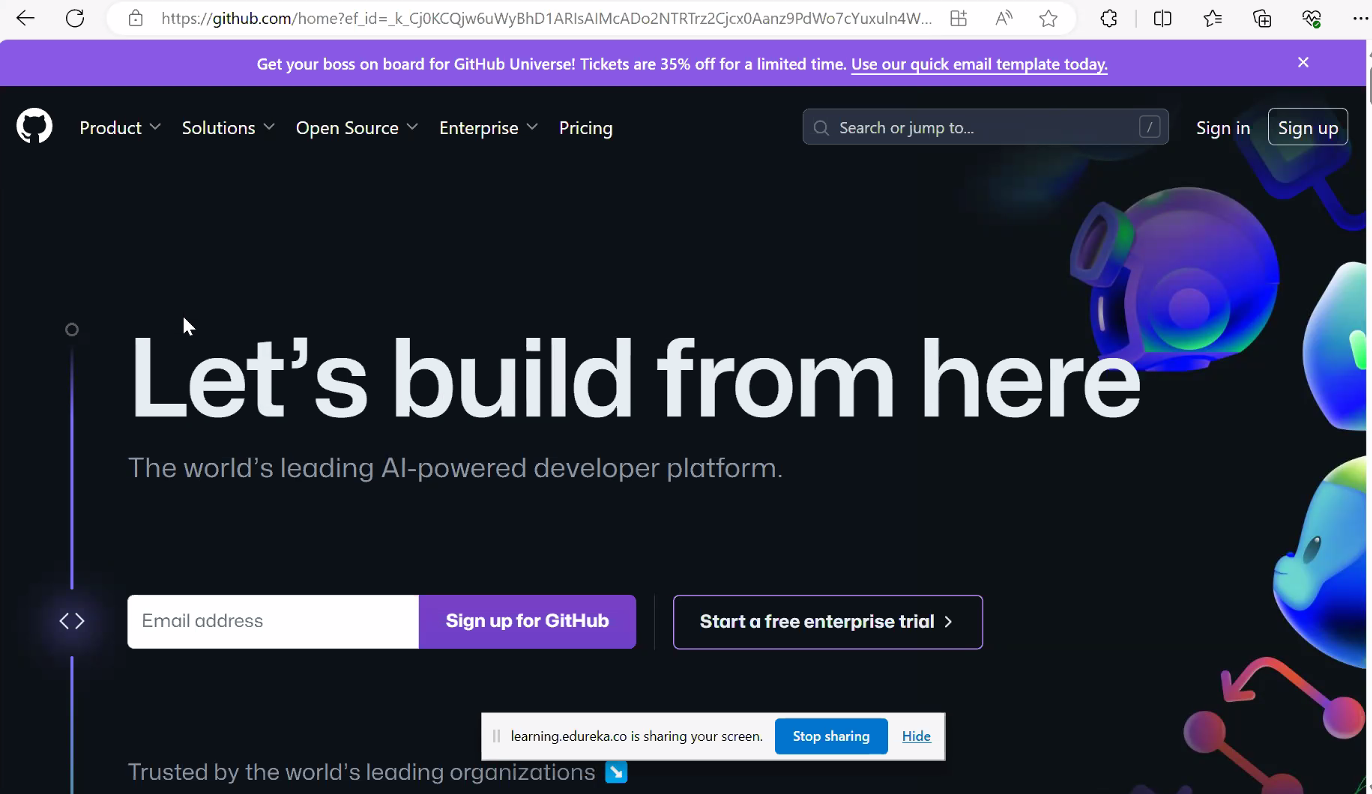
PULL and Fetch

==================================

Demo 1: GITHUB Account creation

To create account in github:

<https://github.com/>



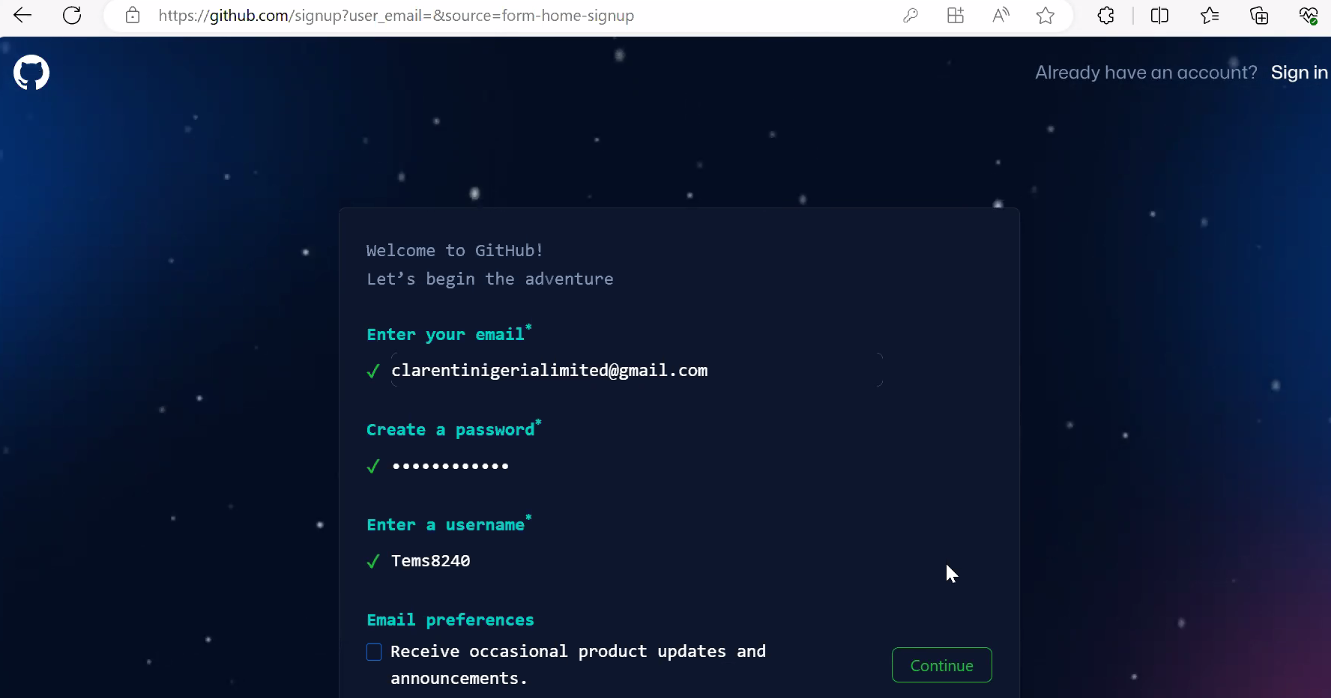
Click on signup

Given valid email address

Give password

Give username

Click on continue.

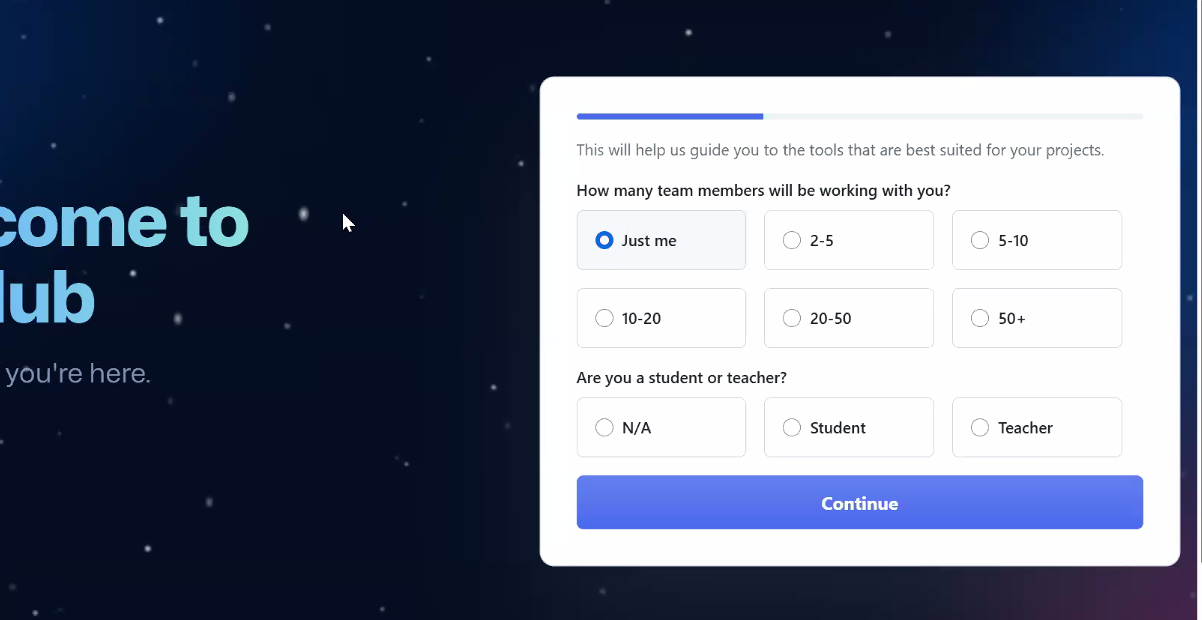


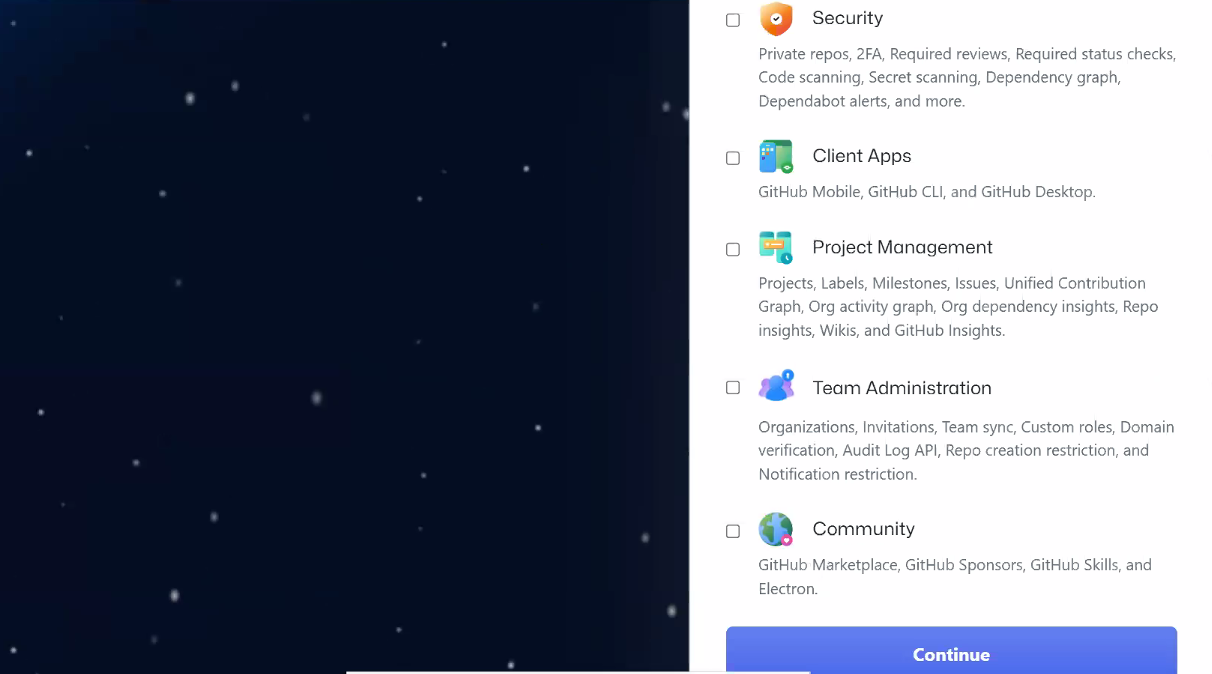
An email will be sent on your mail id

Verify the email

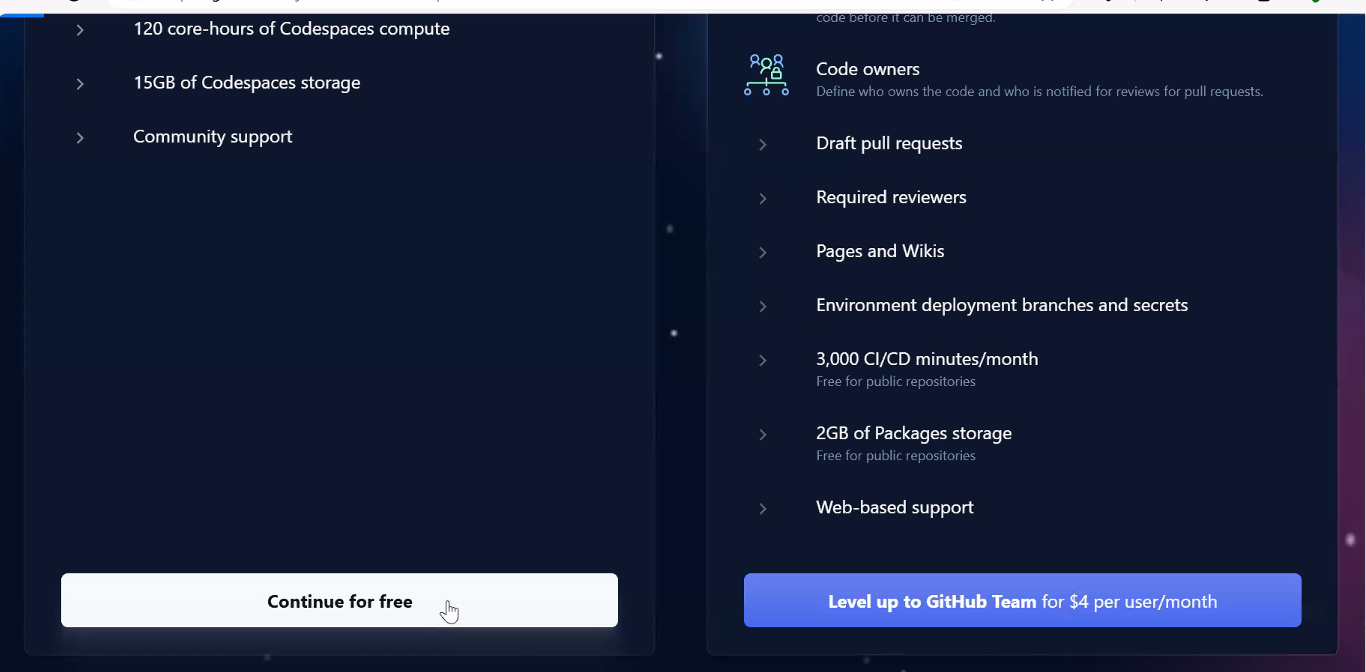
And create the account

> Login to github

****

****

**Continue for free**

****

Demo 2: Create a public remote repository on github

**Working on Remote repository:**

**================================**

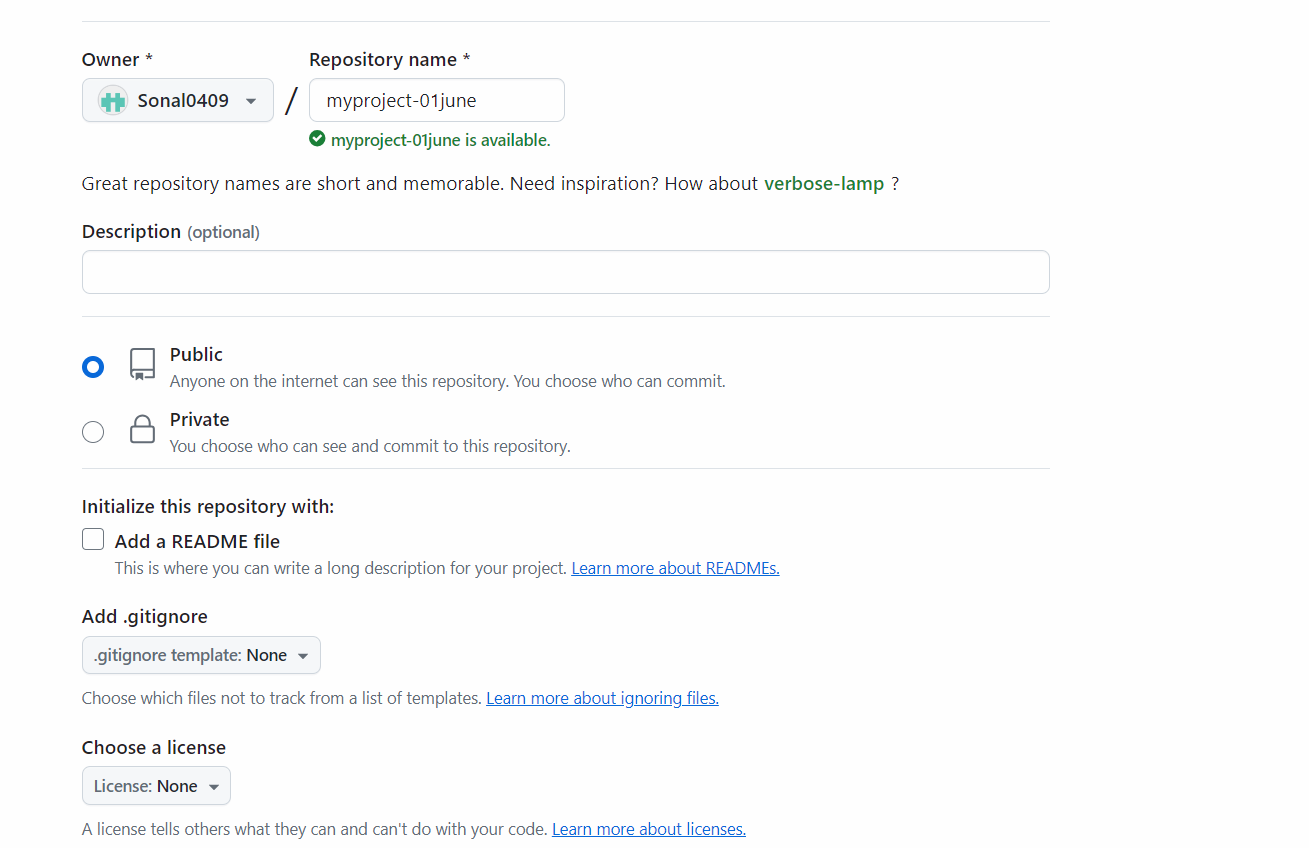
Create a repository:

1. > click on New button to create a remote repo

> Name to the repository -> give any unique name

> select public repo

> click on Create repository

****

=================================

**Create a Personal Access token on git HUb**

=========================================================

**Pls open the below link on a new browser:**

[**https://github.com/settings/tokens**](https://github.com/settings/tokens)

**Go to personal Access token on left side → select Tokens(classic) → Click on generate new token → select classic → you will be on New personal access token (classic)**

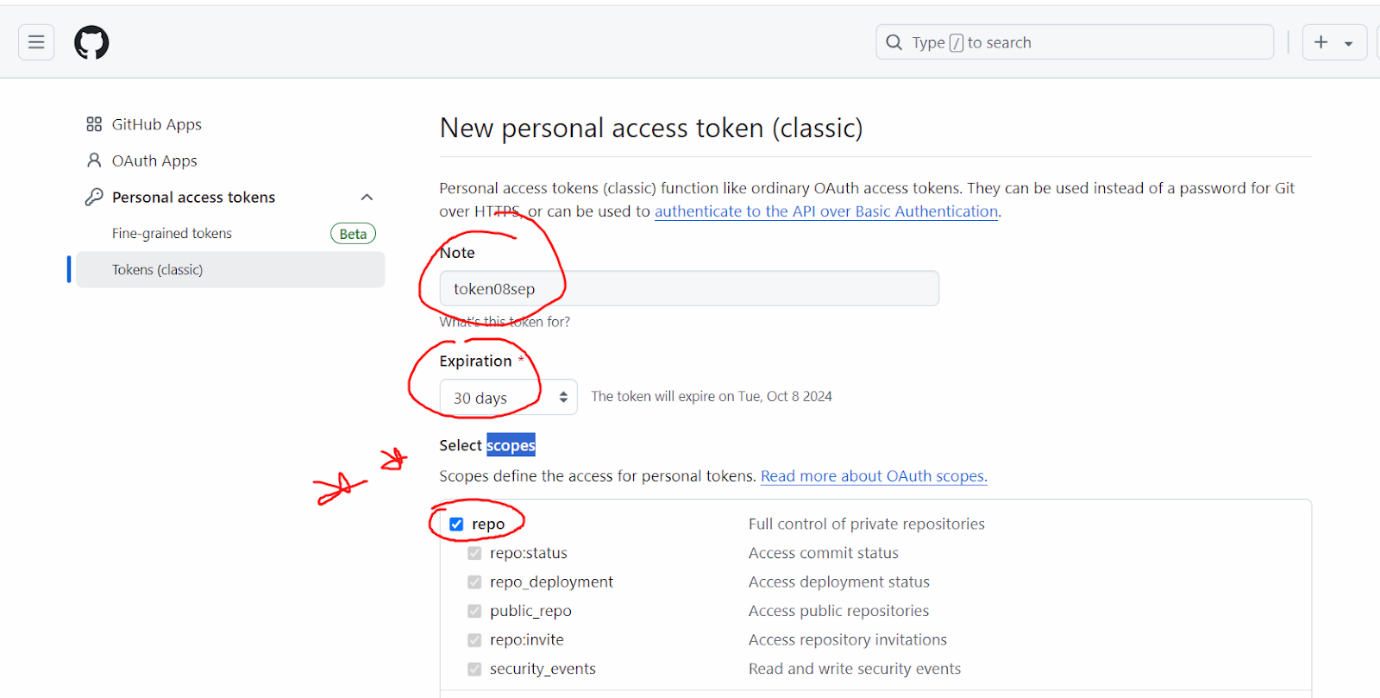
**Give below details**

**Note : give any name to token**

**Expiration: 30days**

**Select scopes: select repo checkbox**

**Scroll down and click on generate token**

****

**Make sure to copy your personal access token now. You won’t be able to see it again!**

==============================

**Connect Local repo to remote repo:**

**=========================================**

**Go to git and execute this command**

**# git checkout master**

**Commit any pending changes**

**# git add .**

**# git commit -m "done"**

**Now connect local git repor to remote repo**

**$ git remote add origin <your remote repo Path>**

**Example like this:**

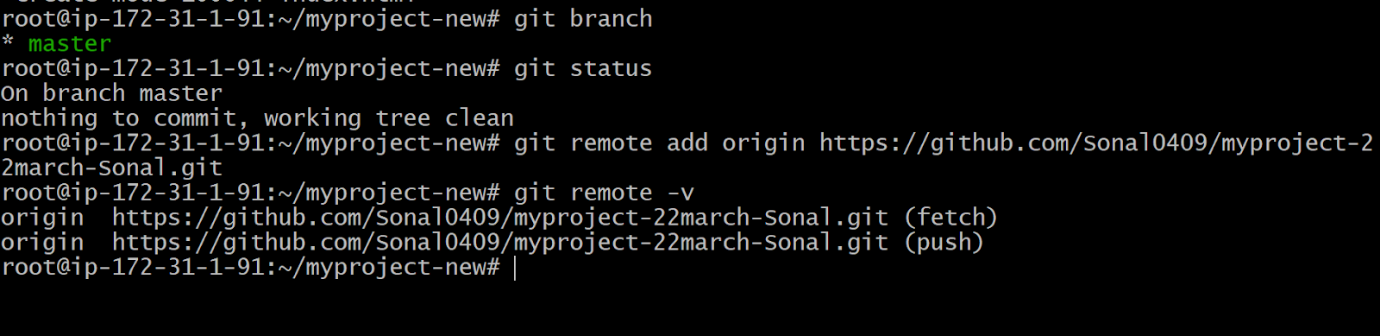
**…**

**git remote add origin https://github.com/Sonal0409/myproject-08sep.git**

**…**

**Execute the command to check local and remote repo is set:**

**$ git remote -v**

****

**PUSH the master branch to remote repo**

**===========================**

**On the remote repo we have no code, no files no commits and no branches**

**# git checkout master**

**# git push origin master**

**Here origin means variable storing path of remote repo**

**It will now ask to enter username**

**Username for '**[**https://github.com**](https://github.com)**':**

**Here paste the token and press enter key**

**Again it will ask for password**

**Password for '**[**https://ghp\_mJ92gregYqNggrggd8AY1NxkWpYigter345Q6PHAM7VxuLg26F4wB@github.com**](https://ghp_mJ92gregYqNggrggd8AY1NxkWpYigter345Q6PHAM7VxuLg26F4wB@github.com)**':**

**JUST PRESS ENTER KEY, do not paste anything for password**

**It will send the files and commits to remote.**

Create files on Remote repo and commit them on Remote repo:

==================================

**Create a new file on the github repo**

**Click on Add file → click on Create new file → give a name to the file → add some content → click on commit → commit on master branch -> click on commit button**

PULL Files from remote to local:

==============================

PULL = FETCH the files +  MERGE with Local repo and working directory

# git pull origin master

# ls

# git ls-files

FETCH:

======================

In this workflow, git will fetch the files but will not merge them to LR or working directory

Create some new files on the remote repo:

==========================

**Click on Add file → click on Create new file → give a name to the file → add some content → click on commit → commit on master branch -> click on commit button**

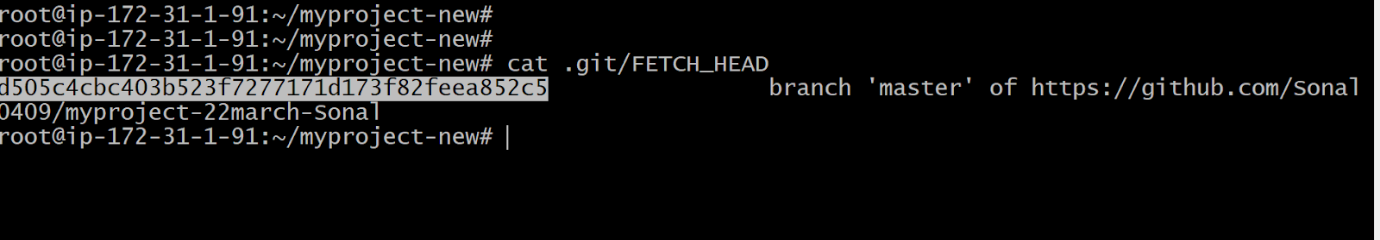
**GO to the git command prompt, execute below command**

**# git fetch origin master**

**YOu want to see the fetched files**

**#  cat .git/FETCH\_HEAD**

**You will see the index number**

****

**COPY the index number that is highlighted**

**See the fetched files**

**# git diff d505c4cbc403b523f7277171d173f82feea852c5**

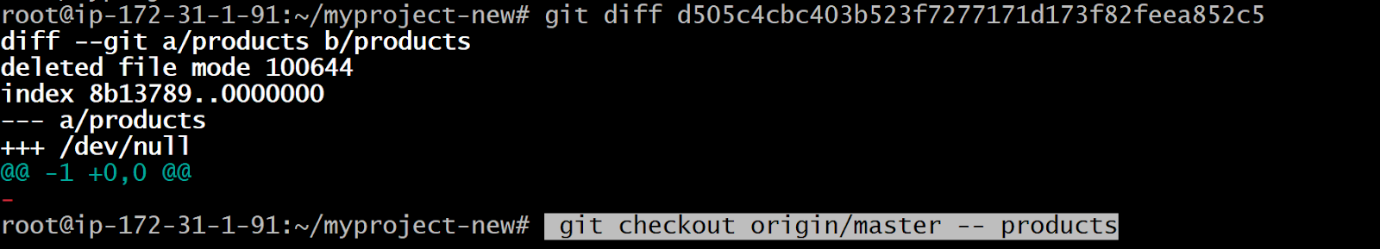
**Merge any specific file:**

**=====================**

**# git checkout origin/master -- <fileName>**

**Example:**

**# git checkout origin/master -- products**

****

**File will be merged to working directory**

**# ls**

**But to merge to local we have to commit the merge**

**# git status**

**#  git commit -m "done"**

**===============================================**

**Clone a github repo to local machine:**

**===============================================**

**Come Out of current folder or directory**

**# cd**

**# git clone https://github.com/Sonal0409/myproject-22march-Sonal.git**

**FORKING**

**=================================**

**A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.**

**operation is performed on github and not on git**

**It is copying github repo to another users github account**

**Open this repo :** [**https://github.com/Sonal0409/myproject-08sep-sonal.git**](https://github.com/Sonal0409/myproject-08sep-sonal.git)

**And click on fork button → new page will open → click on create fork.**

**Pull Requests:**

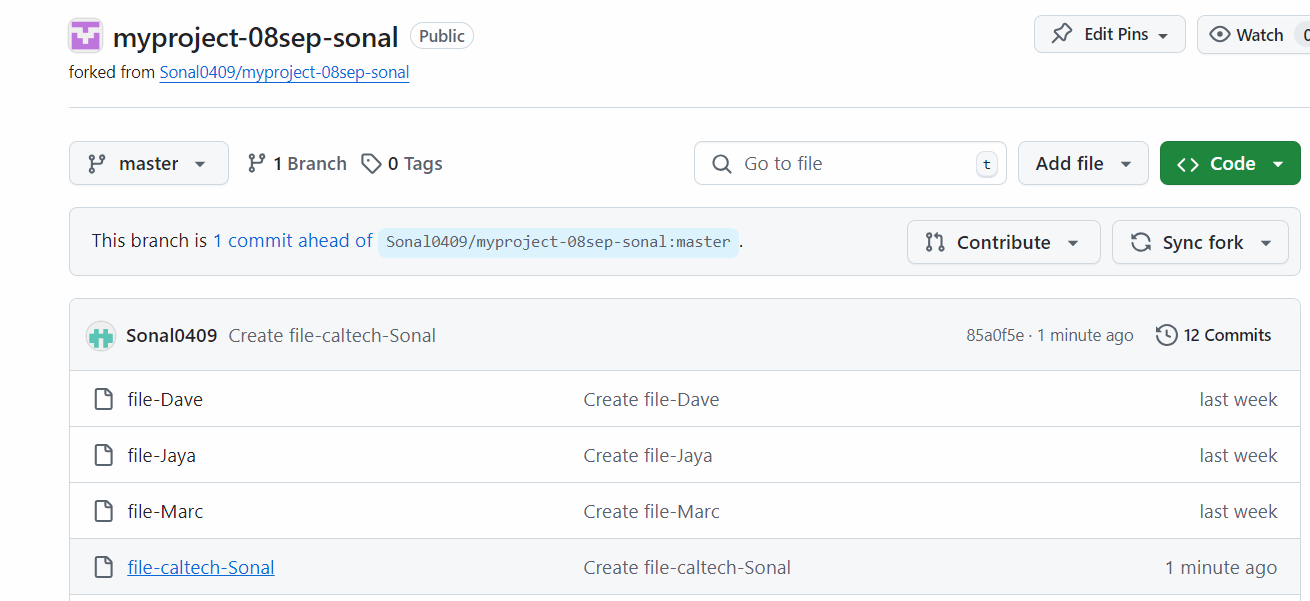
**=====================**

**Fork the below repository:**

[**https://github.com/Sonal0409/myproject-08sep-sonal.git**](https://github.com/Sonal0409/myproject-08sep-sonal.git)

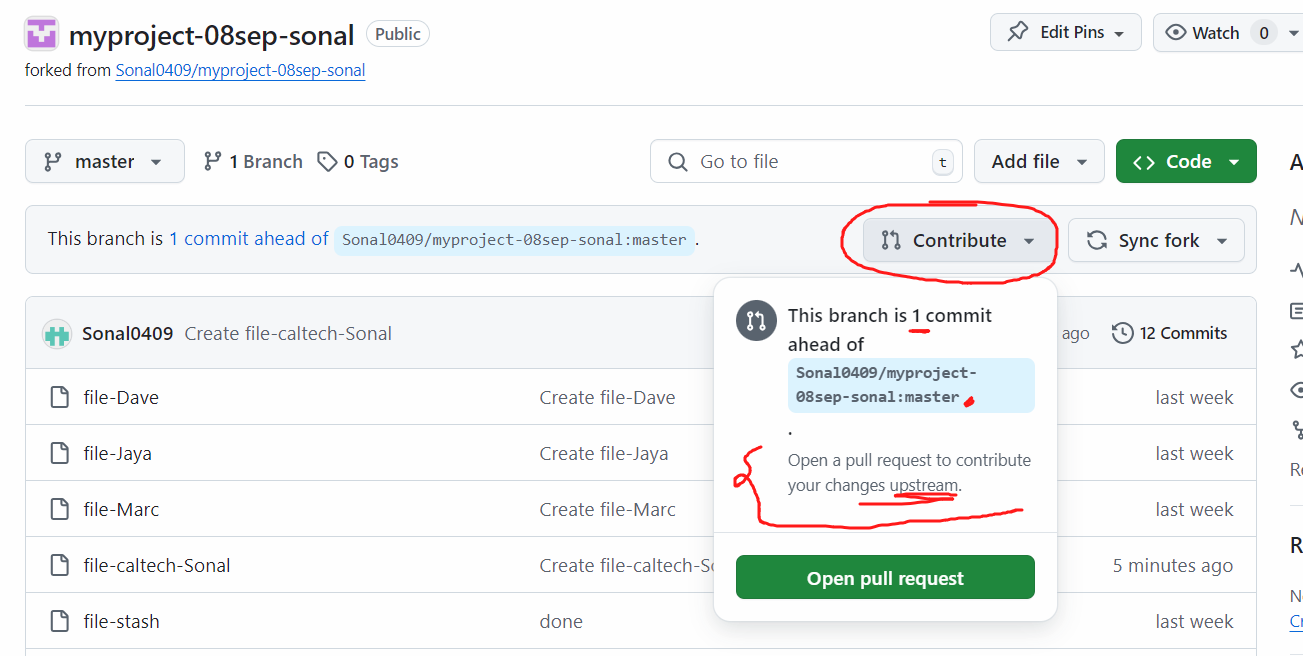
**You will have a copy of the above repository in your github account.**

**Now create a file on your copy of the repo**

****

**Now we want to contribute this file to the parent repository**

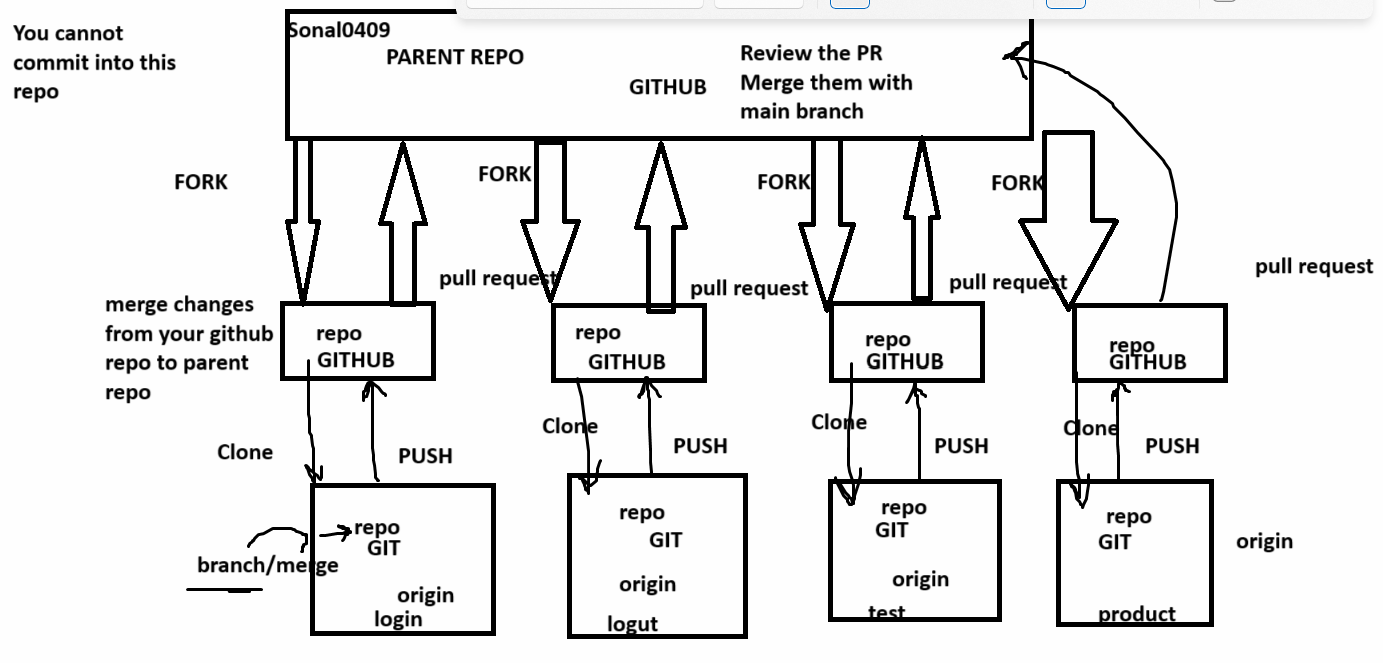
**Click on Open pull requests button**

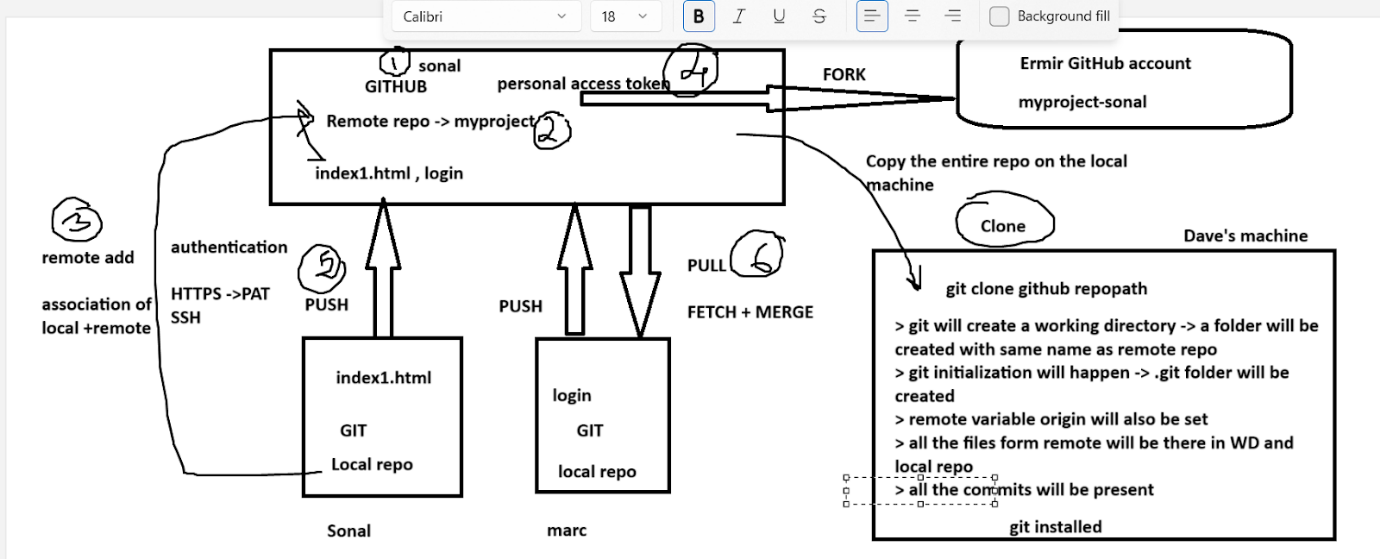
****

**Open a pull requests for the parent repo**

**The owner will now merge the pullrequests on the parent repo**

**Now you can click on Sync fork button and click on update branch→ to get new changes form parent repo to your copy of the repo**

****

****

**Day 6 : 23 march**

**========================**

Conflicts and resolve conflicts

===================================

We may get into conflicts when we are working on same file

In the file if there are changes that are not available in my copy of the file, I will get conflict.

How to resolve the conflict

You can resolve it by manually adding the changes and commit them

Conflicts can occur on github and on git

Conflicts can occur when we are working with merging branches of same repo or different repo

Merging of branches is done using PULL request concept

When we raise a pull request github shows us if branches can be merged or there are conflicts

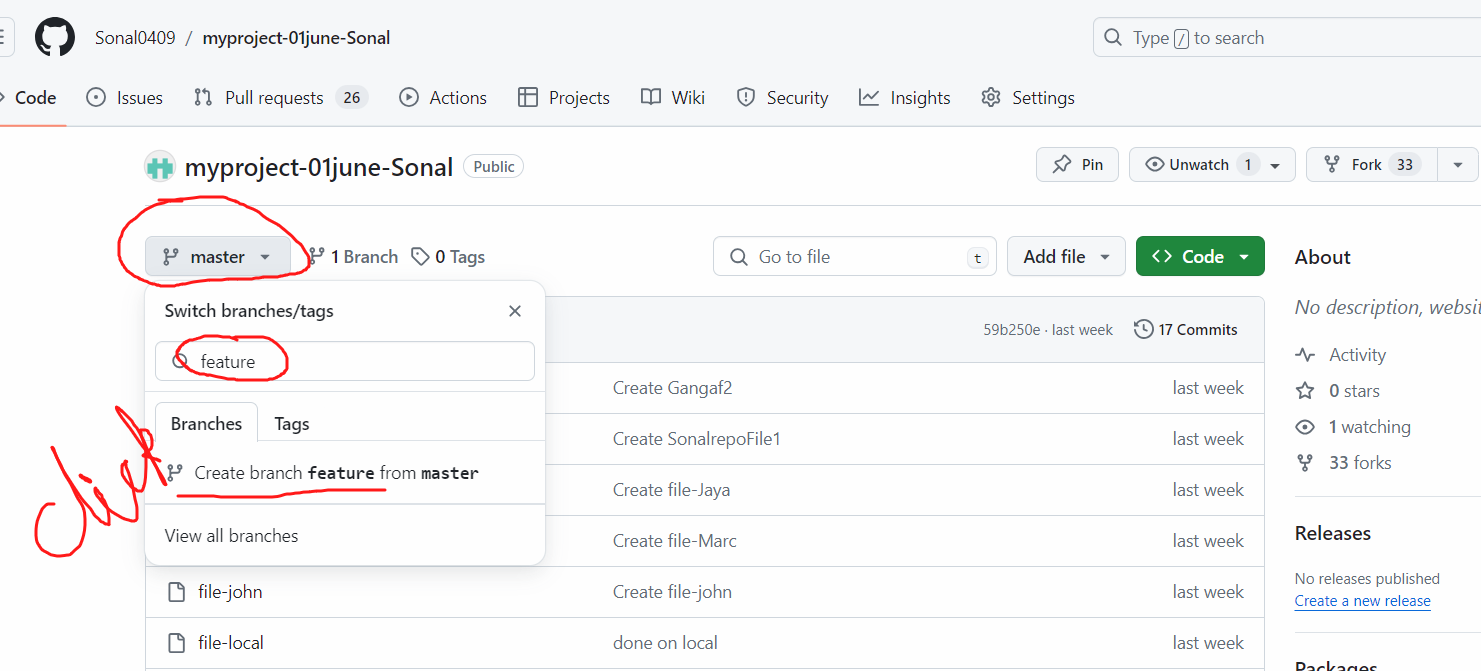
Conflicts and resolve conflicts in GITHUB

======================================

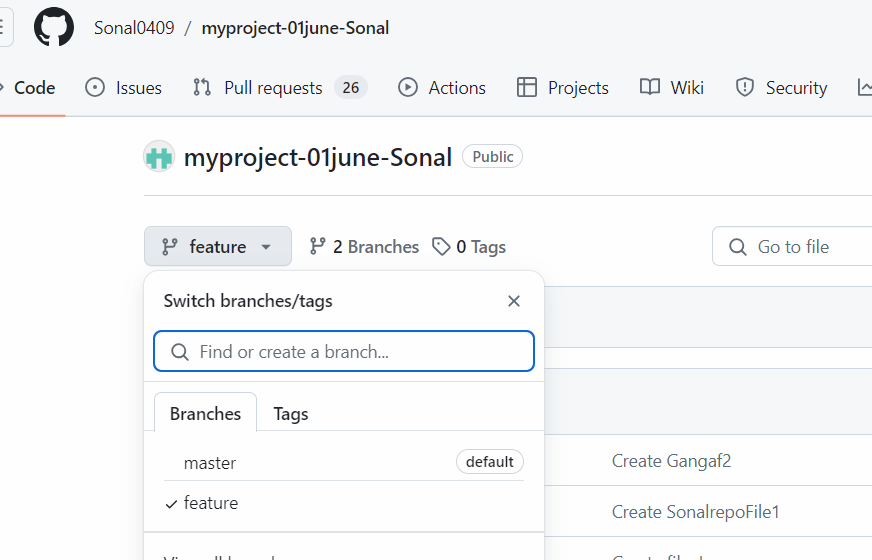
Go to your github repository

The repo has some files and a branch Master

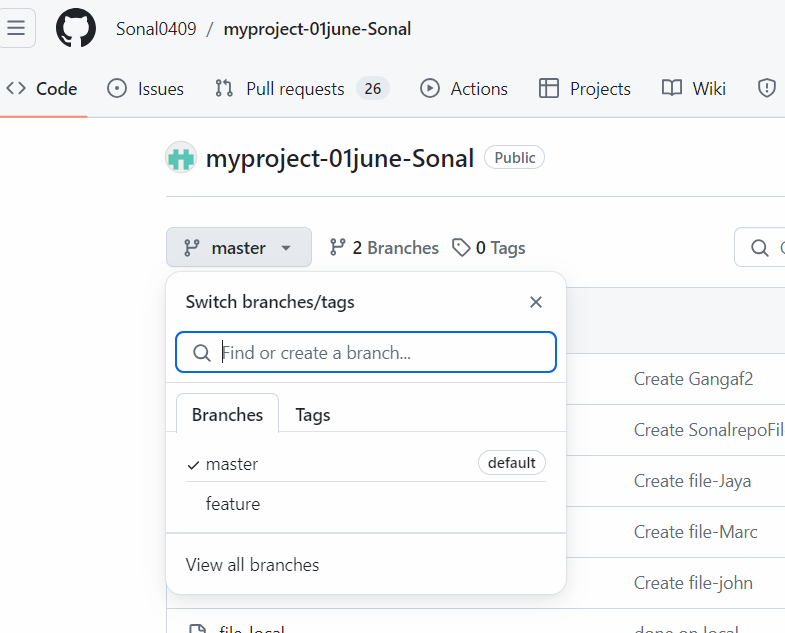
Now create a new branch on the repository



You will have 2 branches now

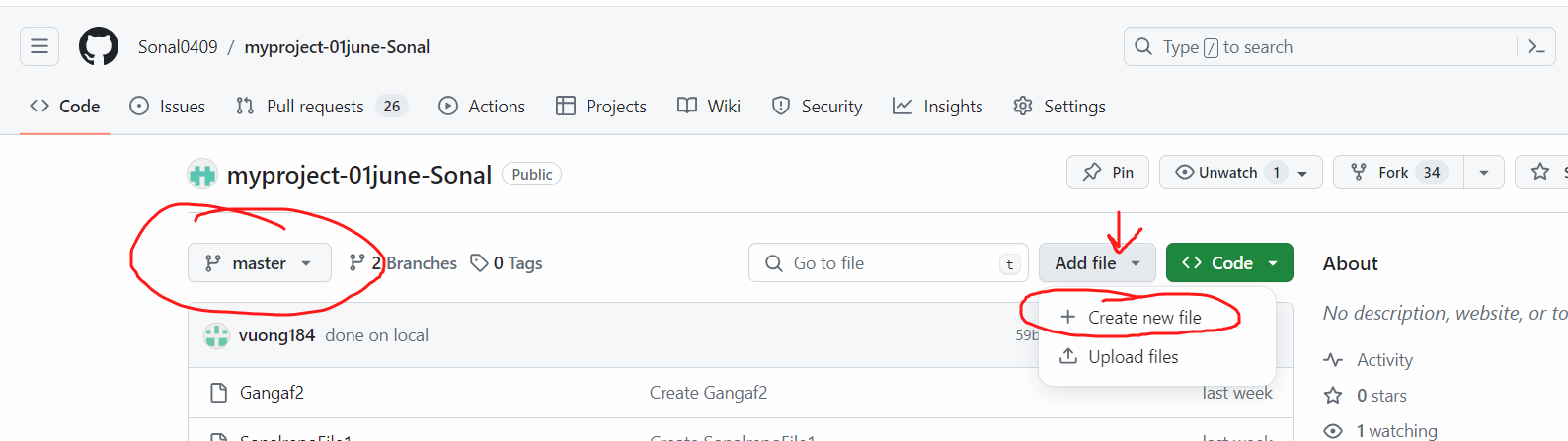


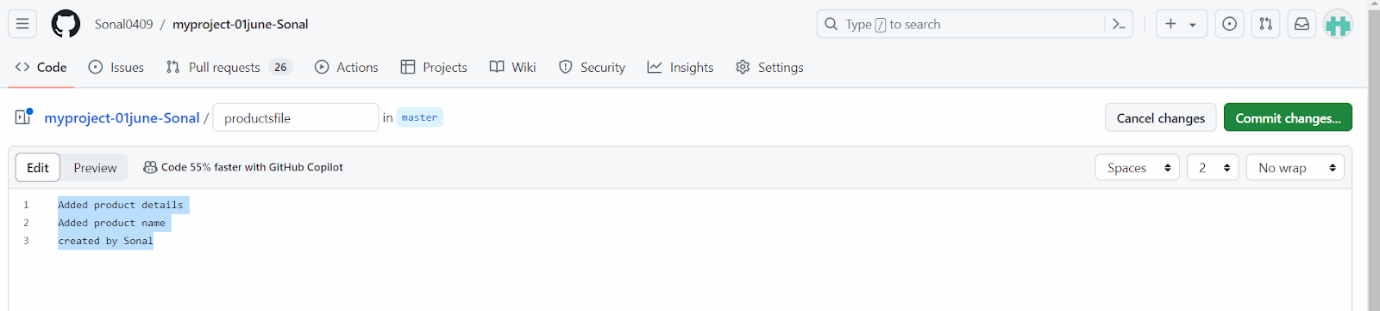
Click on master branch

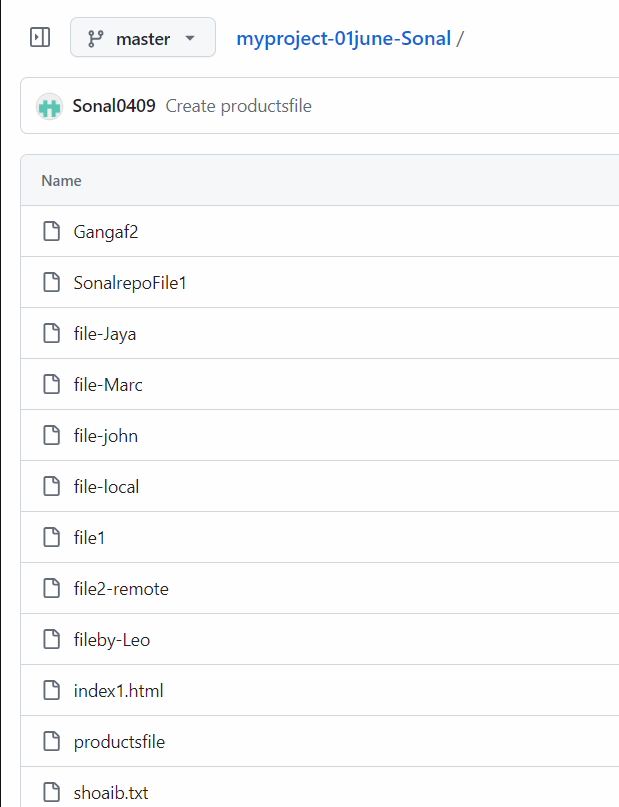


Create a new file with name as productsfile on the master branch

Add few content inside the file

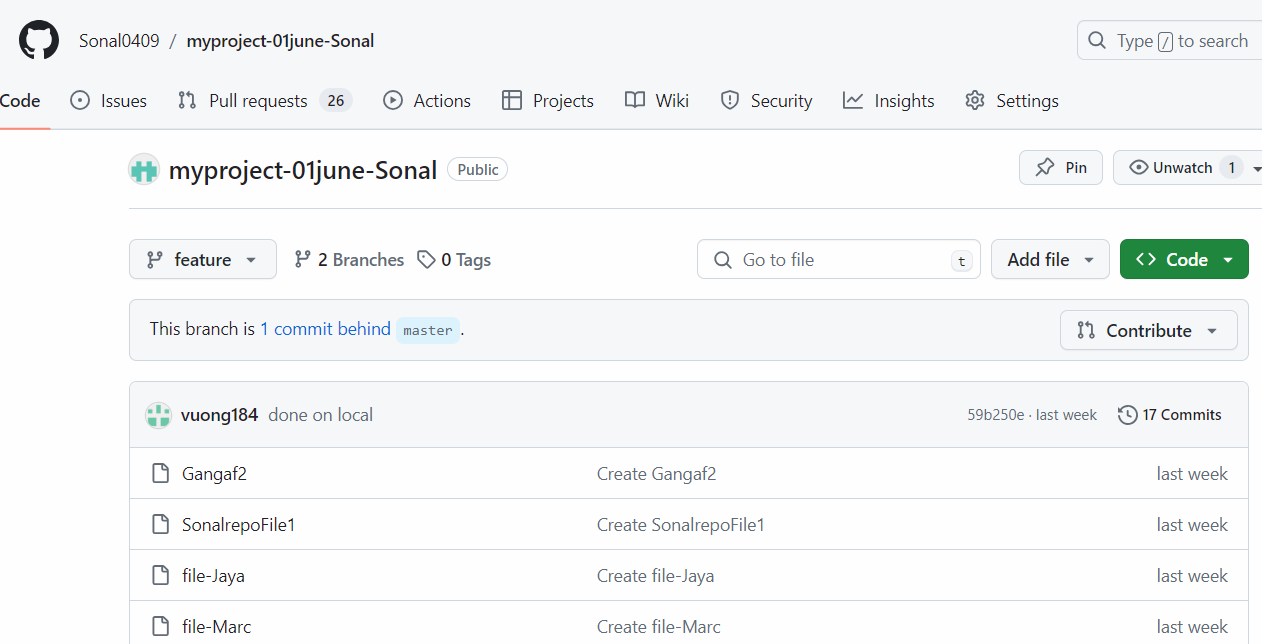






This file is not present on the feature branch

So we will switch to the feature branch



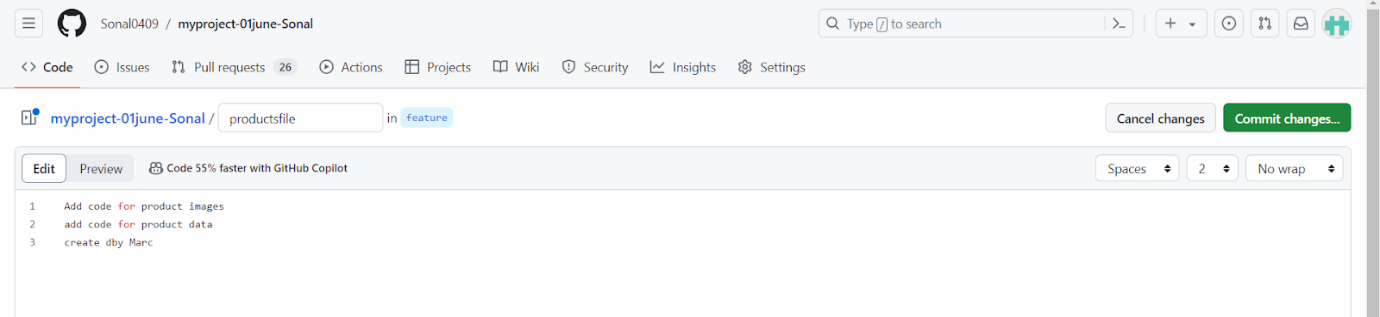
Lets say we have to work on products file.

On this branch there is not productsfile

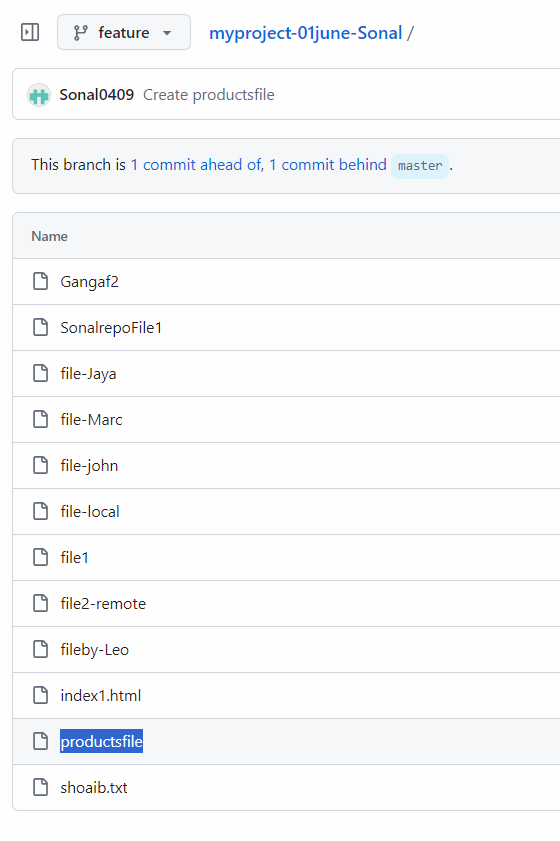
So we will create again a file with same name as productsfile

But in this file we add different content which is not there in productfile on master branch

So we have created a real time use case where we have 2 branches with same file name but different content contributed by different users.

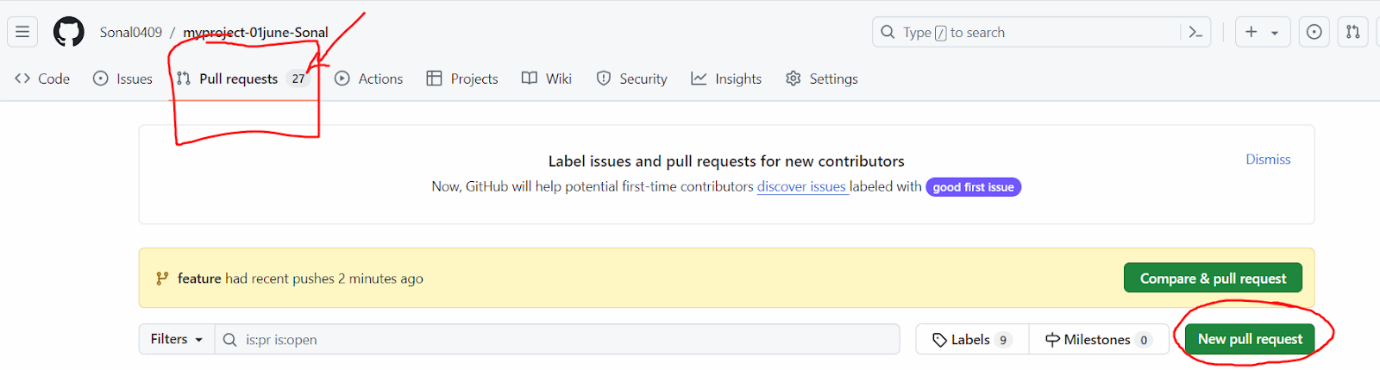


Commit the changes



Now lets us merge feature branch to master branch

In github if we have to merge we have to raise pull requests

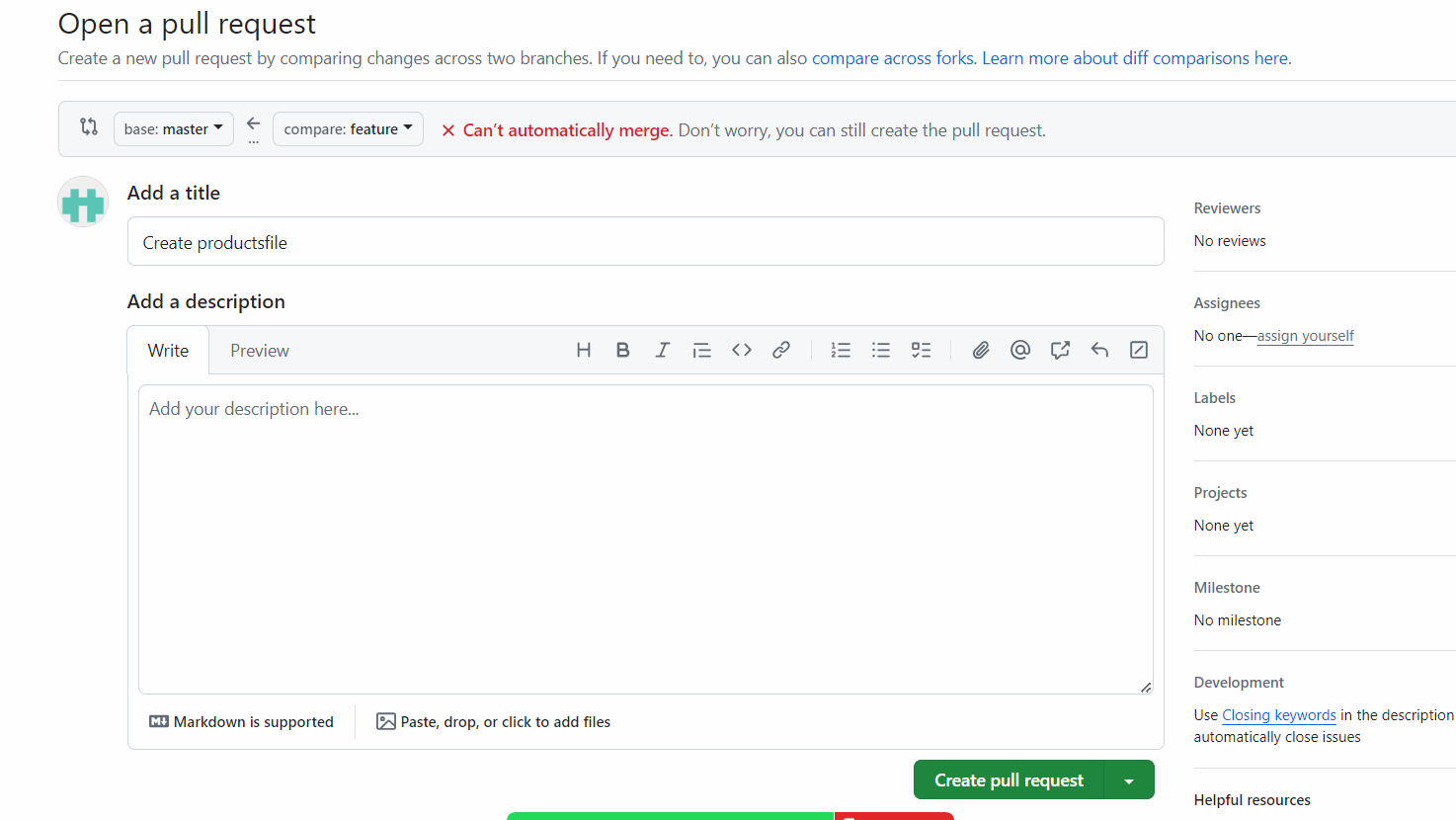


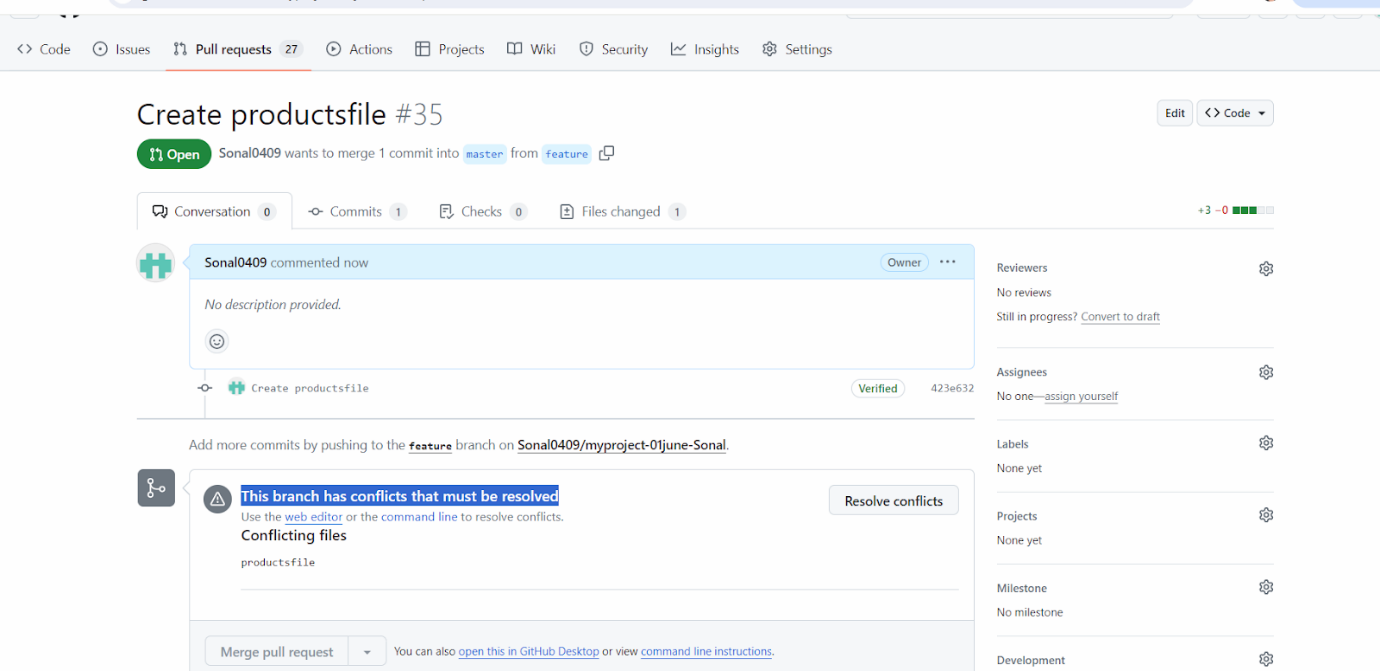
As we raise the pull request we see that the 2 branches have conflicts



However we can still go head and raise the pull request

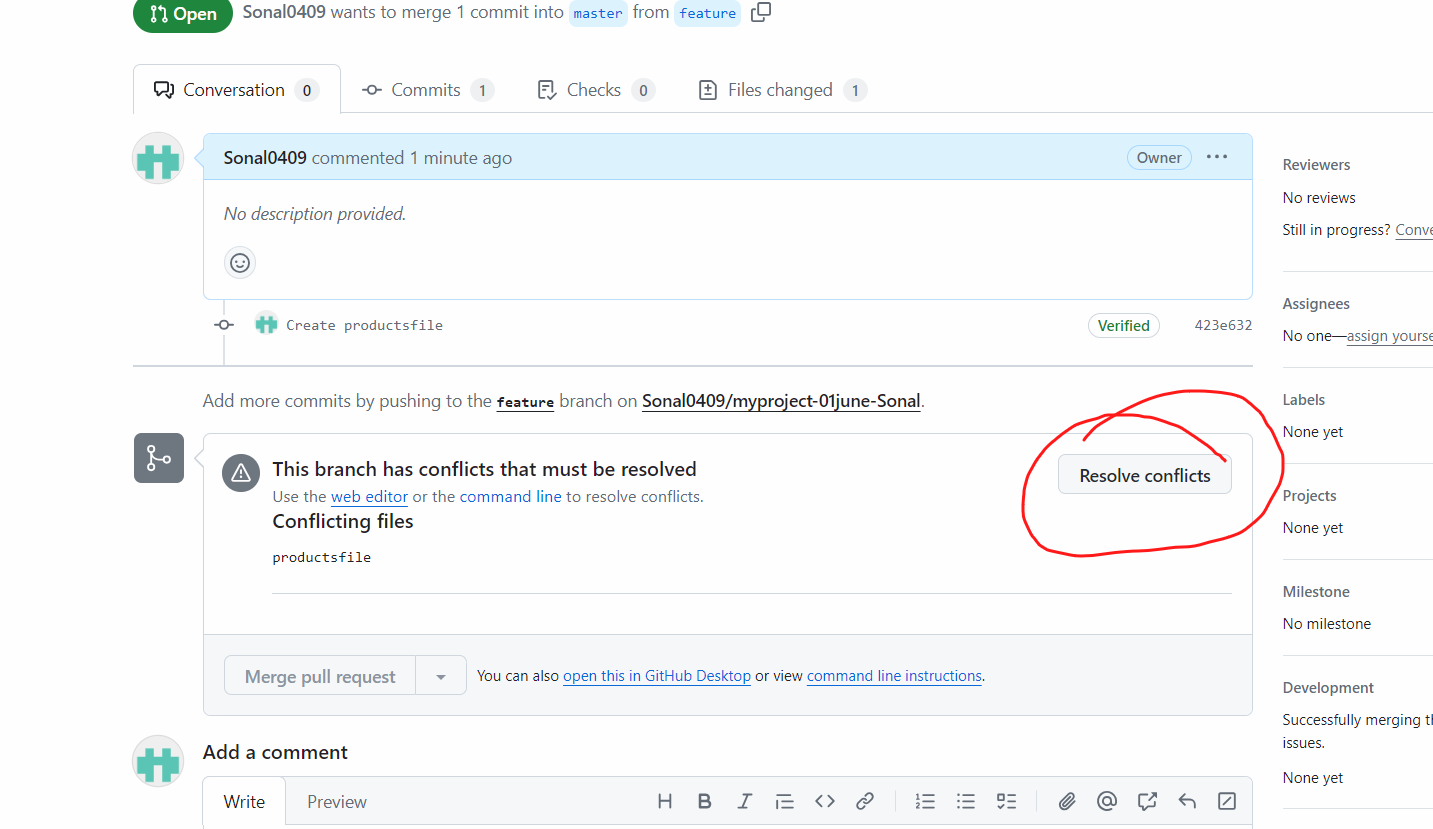
Once the pull request is raised we can resolve the conflicts



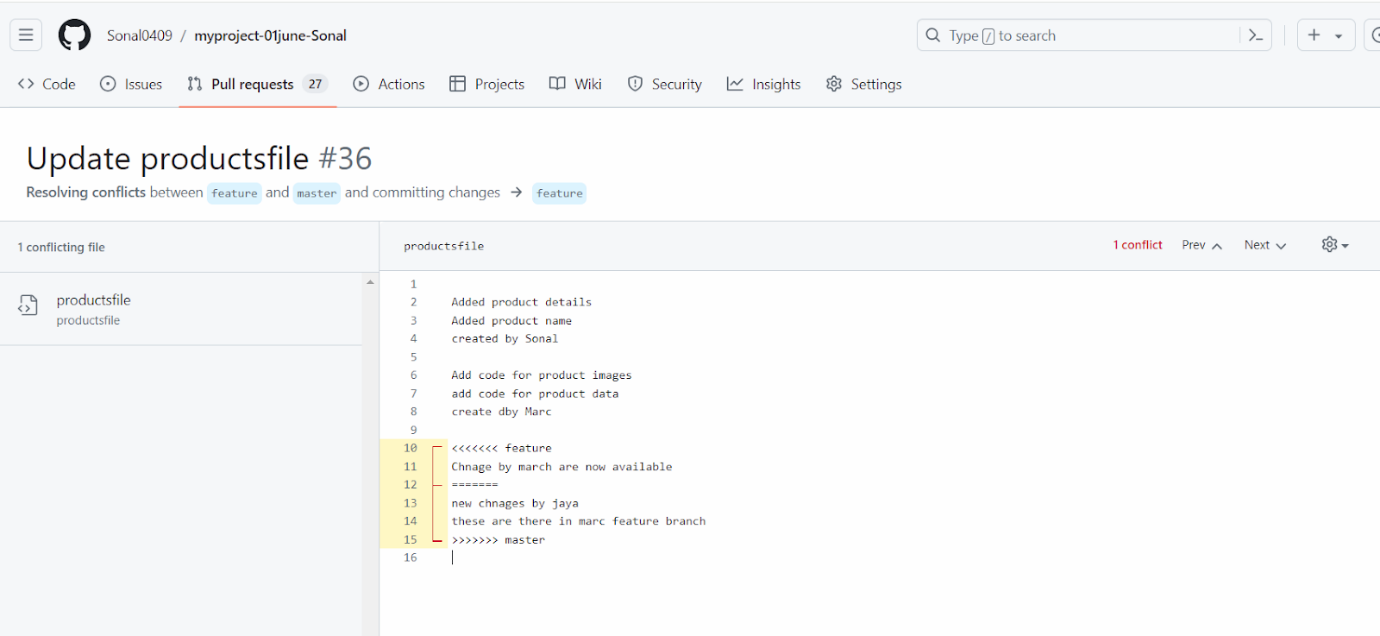


Resolve Conflicts

Click on resolve conflicts button

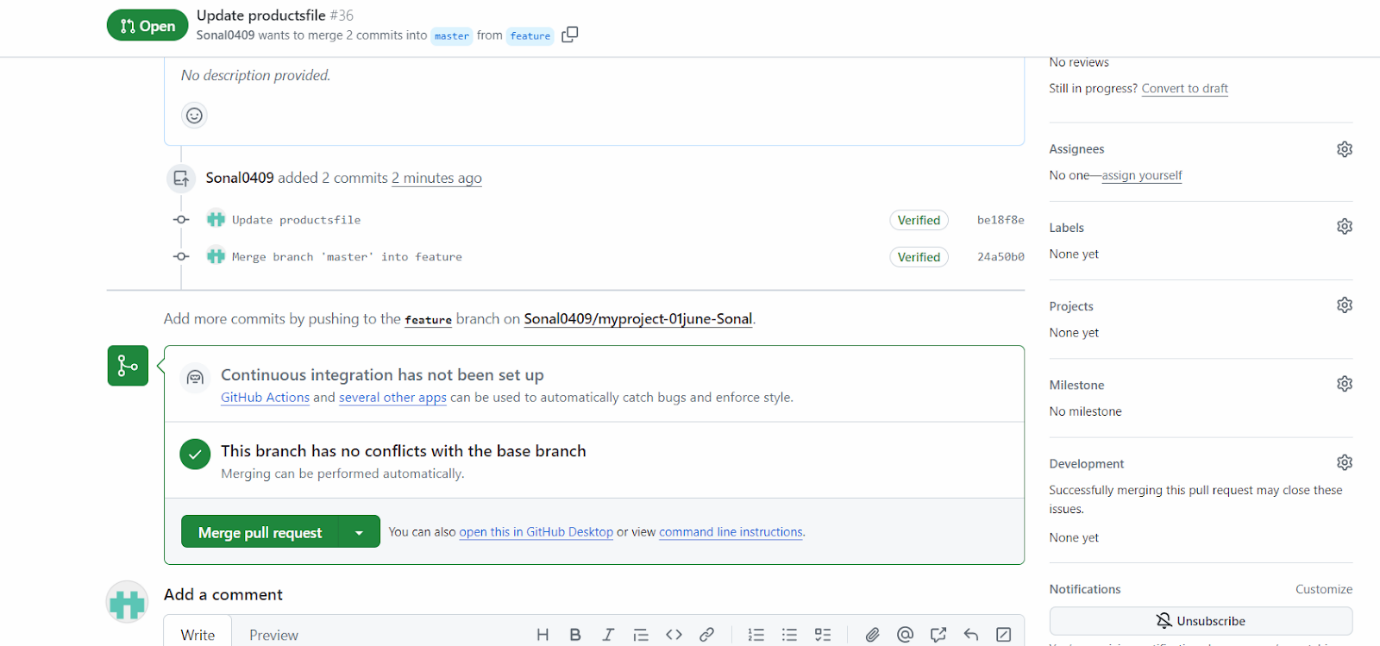


Remove the extra lines like ==== or >>>>



Commit the merge after this

And click on merge pull request



==================================

Change to root user:

**# sudo su -**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Install Jenkins:

===================================

On the browser open **https://www.jenkins.io/**

Go to downloads section

Select centos OS

Go to Ec2 machine and execute commands:

# sudo su -

# apt-get update

# apt install openjdk-17-jdk -y

Install jenkins

===========================

# sudo wget -O /usr/share/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

# echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list **>** /dev/null

# sudo apt-get update

# sudo apt-get install jenkins

# systemctl start jenkins

# systemctl status jenkins

We need to now  set up jenkins dashboard

=======================================

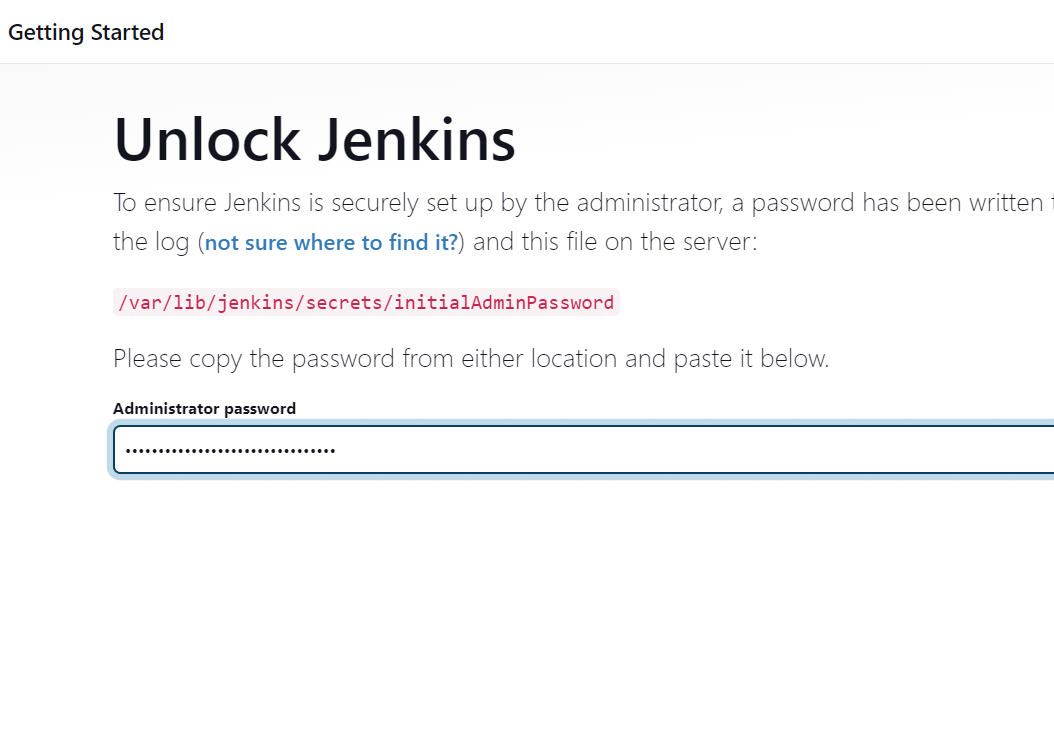
For this, take public ip address of ec2 server, copy it and go to your browser and give

http://publicipaddress:8080

On the ec2 server terminal , execute below command

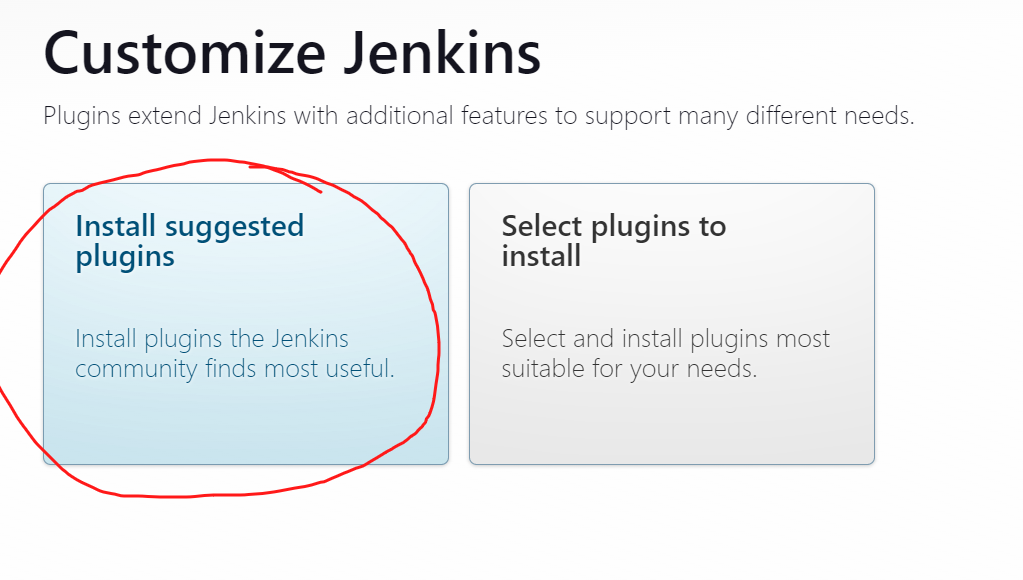
$ **cat /var/lib/jenkins/secrets/initialAdminPassword**

Copy the password and paste in the browser (jenkins)



Click on continue

Click on Jenkins suggested plugin tab(on left side)



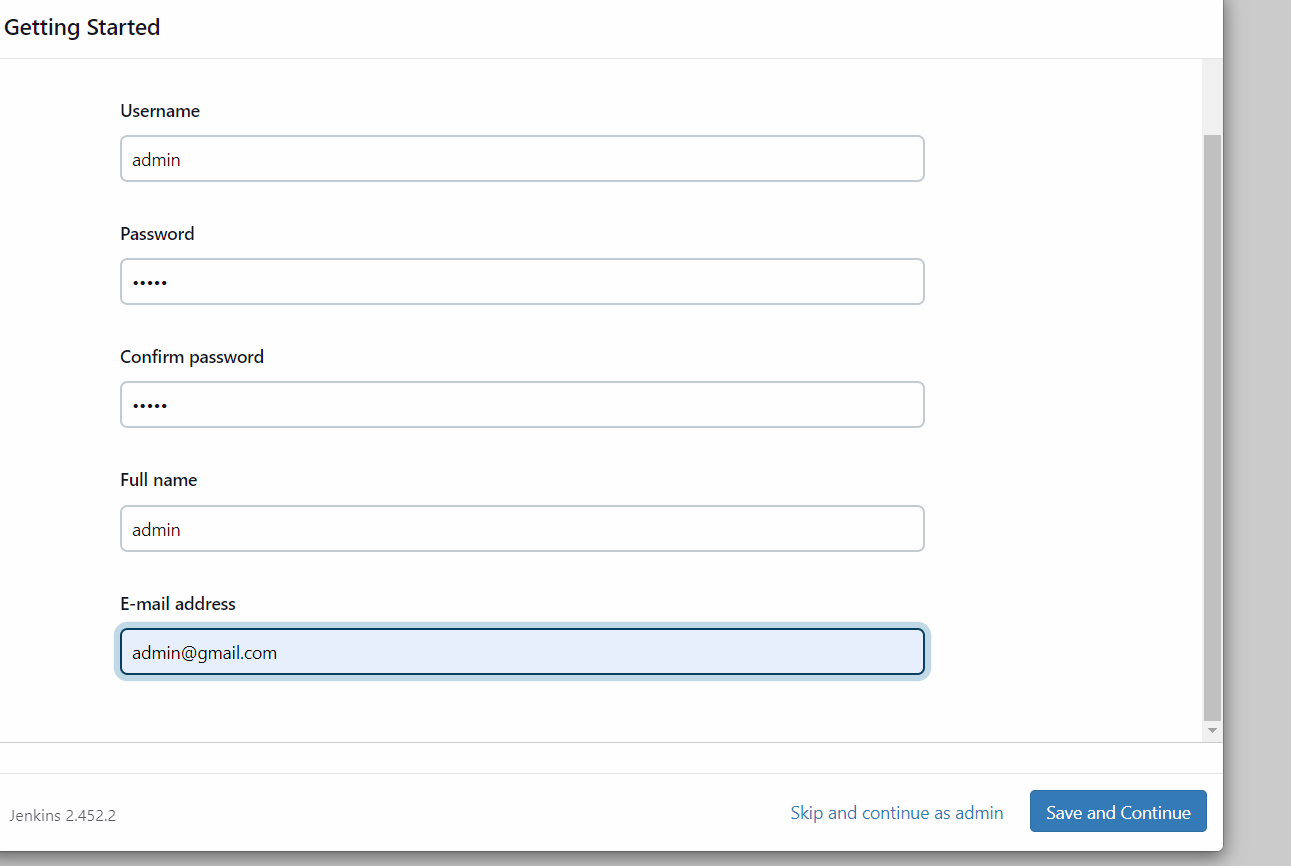
On the next page

Username: admin

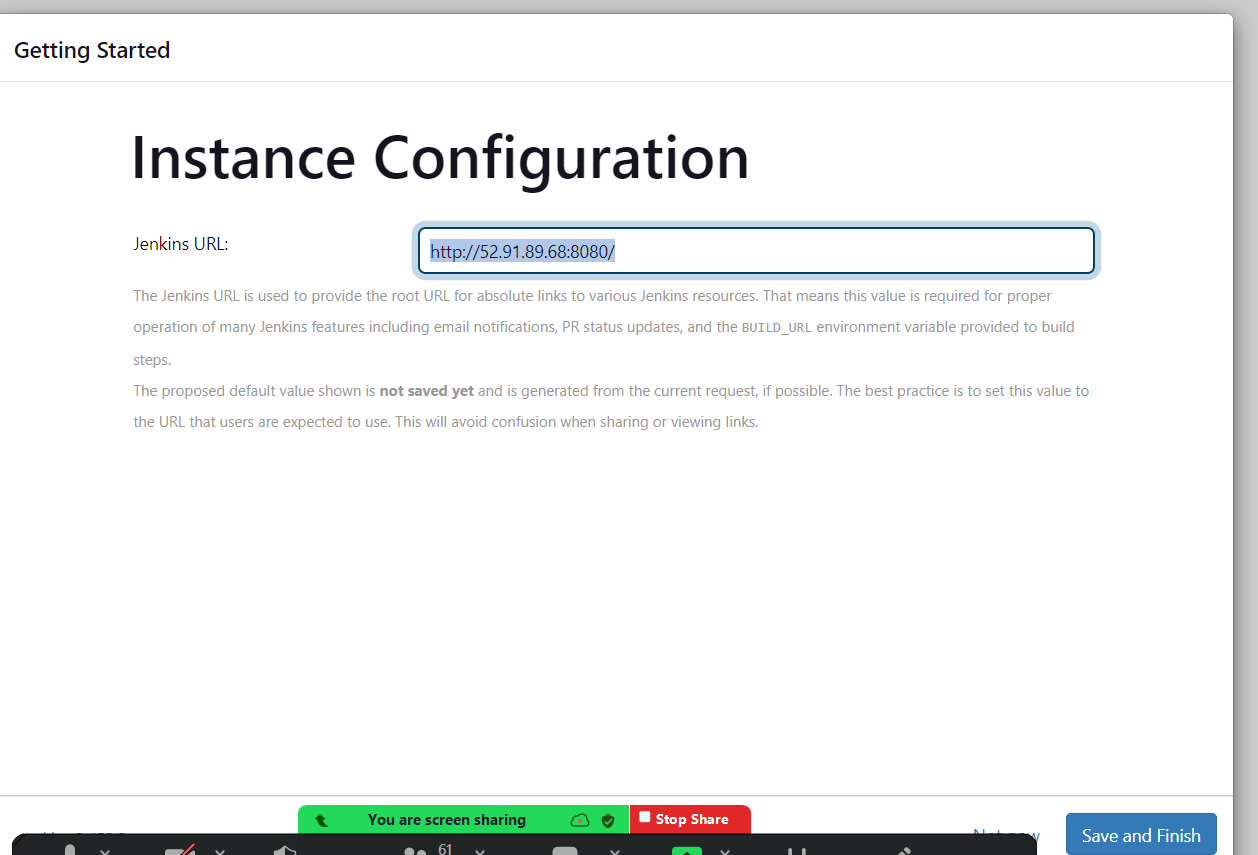
Password : admin

Retype password: admin

Email: [admin@gmail.com](mailto:admin@gmail.com)



Click on continue



Click on finish

You will be on the jenkins dashboard

Demo 1:

======================================

Create a Jenkisn job that will execute linux commands.

Create a new job in jenkins

Click on + sign to create new item

Give a name to the job : Job1

Select freestyle project and click on OK button

Go to **build Steps**→ select **Execute Shell**→ give commands like

echo "Hello Jenkins"

touch file1

Save the job → click on **build Now** button.

Click on the **build Number** → click on Console output

Click on Job1 name→ to come back to job dashboard

See the console output and check the workspace directory.

===========================================

JENKINS INTEGRATION WITH GIT AND GITHUB

===========================================

Demo 2: Create a Jenkisn job that will clone a github repository in jenkins workspace

Create a new job in jenkins

Click on + sign to create new item/job/project

Give a name to the job : CloneRepo

Select freestyle project and click on OK button

On the project click on Source code management

Select git option

Give git hub repo path

https://github.com/Sonal0409/myproject-22march-Sonal.git

Branch name leave it as it is===>  master

Save the job

Click on Build now

Repository will be cloned in jenkins workspace

You can go to job → workspace folder to see the files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For you information:

Create a new job → general section→ click on advance → custom workspace

Give directory as /tmp/myworkspace

Save job → build

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**For you to practice latter**

Wipe out the workspace

On the job → go to build environments → Select first checkbox →Delete workspace before build

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Jenkins Triggers

=============================

     To automatically execute a jenkins job, you will use Jenkins triggers

       Based on:

      Periodic time- schedule

      Whenever there is a commit in github repo

      A parent job can trigger execution of child job

      Execute a job from remote script

You will go to build triggered section in the job to set up a trigger for automatic execution of the job

Trigger periodically

============================

Cron Syntax:

MINUTE HOUR DOM MONTH DOW

MINUTEﾠMinutes within the hour (0–59)

HOURﾠThe hour of the day (0–23)

DOMﾠThe day of the month (1–31)

MONTHﾠThe month (1–12)

DOWﾠThe day of the week (0–7) where 0 and 7 are Sunday.

USE this if required: [https://crontab.guru/#15\_\*\_\*\_\*\_\*](https://crontab.guru/#15_*_*_*_*)

Demo:

Go to Job Clone repo --> build trigger--> build periodically →

 give \* \* \* \* \* --> Save and build now

All build will be generated automatically every 1 mins

Demo:

Go to the Clone Job → go to triggers

Build Triggers--> Select Poll SCM -->give schedule as → \* \* \* \* \*

Build will be generated each time there is a change in the repository only.

===========================================

Webhook:

Webhooks allow Jenkins to be notified when certain events happen on the repo. When the specified events happen, we’ll send a POST request to each of the URLs you provide.

Go to jenkins Job clone repo→ go to Trigger s → select  GitHub hook trigger for GITScm polling

Save the job and now go to github

Always save the job and then only  go to github, perform these steps

Take any of your repositories—>go to the repository settings

select webhooks on left side

delete if any existing webhook

click on create webhook on right side

provide following information:

Payload URL : jenkinsurl/github-webhook/ ===> http://3.140.252.165:8080/github-webhook/

Content type : select application/json

Secret: no need of any value

Which events would you like to trigger this webhook?

select first option -- Just the push event.

Select Active

click on add webhook.

now make some changes in repo,

go to jenkins--> you will see a new build has been created