**GIT Class 1**

**PROBLEMS**

**CASE STUDY 1 :** **SLIDE-3**

* You are writing java code, to make a calculator. You completed the code and saved it. Your calculator code currently can do simple operations like + , - , \*, /.
* Next day you wake up and think of an idea to implement %, square, square root in that code
* You change that code but there is some error now and your calculator doesn't work.

Your friend asked you for the basic calculator code .

how will you go back to the previous code you have written?

**Solution 1 :** **SLIDE-4**

Make a file on computer and show the solution of making copies

***a)Make different copy of codes and edit****:*

One of the solutions is to make a copy of the code and then edit the copy, but suppose that it's a very big code and you will need to make changes in future also.

How many copies will you make?

What about the space on your harddisk?

How will you keep track of changes made in each code?

**What if size of the project is big**

**What if we need to change something and we need to go back**

**Solution 2:**

***b)GIT****:*

So in order to solve this problem one day a person like you guys who was making copies of changing code thought of resolving this problem to make life easy for him and others. So he came up with a fantastic solution called GIT.

But before we discuss what is git lets quickly look into the concepts of versions.  
So what are versions? **SLIDE-6**

**Git a Version Control: SLIDE-7**

* It is a version control tool
* Every change we make in a code anytime and save it , it makes it a version of that code.
* If i am writing code in java and i write a code for a calculator that can do + ,-, /,\* and I saved it. → This will be version--1 of my code
* Next day I make some changes and add square root, % in my code and save it .--> This will be version 2 of my code.

So we can say that

* Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.
* Version Controls keeps track of all the version of our project, every small change creates a new version of the project.
* VCS Keeps multiple (older and newer) versions of everything (not just source code).
* VCS displays differences between versions

**Popular VCS:**

The tool which handles all the backups are called as Version control tools. Most popular vcs tools:

* GIT
* SVN
* CVS
* Bazaar
* Mercurial

**SLIDE-8**

**Now the problem which we discussed earlier is solved using git**

If I need to go back to my version 1 code it's already saved by GIT and i can easily go back to it. **SLIDE-9**

GIT write messages and save our version with date and time:

In git we can also write messages of the changes we have made in each version and the date and time on which we did changes so its very easy for us to keep track of our versions and changes that we have made in a code

CASE STUDY 2: **SLIDE-11**

***a)Laptop fails****:*

* Now everything we do is on our computer!
* Let's assume one day John was working on his code and his laptop just stopped working.  
  His all data is lost. What will he do?
* In order to have his data saved for the future he must save it also somewhere apart from his computer.

Solution:

In this scenario we have a website called GITHUB.com that allows us to upload our code on it and if any problem happens with our computer our code is still safe.

**GITHUB:**

For now just know that git hub is just the copy of your local repository on the internet , in future we will discuss more in depth about the it

CASE STUDY 3: **SLIDE-12 SLIDE-13**

A user story and there are 3 testers available

* To do the work quickly tasks are divided it into 3 parts assigning each one of the tester a unique task
* Using github everyone can review progress .check the scripts and add comments and hence everyone can work in collaboration

**------------------------------------------------------------------------------------------------------------**

* We learned why do we need Git and Github and how it makes our life easy and helps us work together in collaboration
  + Git is just one of the VCT, there are other popular SVN
  + Why to use GIT? —> to collaborate with teams
  + When you work in a team on the same file/project —> it is very helpful and useful
  + Git is an open source

**SLIDE-14**

**Command Terminal: SLIDE-15 SLIDE-16**

What is a terminal?

* So now in order for us to learn how we can send commands using our git we need to look into few CMD Commands.
* On the Mac, the command line is a program called **Terminal**.
* In windows, you install your own command line program - **GitBash**.

Windows comes with a command line, but it is non-standard and more difficult to use.

Windows user will be using **gitbash** and mac will be using **terminal.**

**In there we will type the commands that are used to actually interact with git and use it for the purpose of Version control**

But First we need to learn how to navigate from 1 folder to another folder. **but why**?

**Use case: (Explain in more detail that why do we need in actual these commands for navigation)**

**If we have a code in our folder in the c drive inside some folder xyz and now we want to use git to keep trach of it, first we need to move our terminal into that folder**

**Once our terminal is inside that folder then we can use the git commands**

This commands that we will be writing are very powerful.

In order to use GIT we must be familiar with few basic terminal commands that will help us to navigate into different directories

Let’s open terminal:

Now let’s just type

* **whoami**
* While you're logged in to the system, you might be willing to know : Who am I? The easiest way to find out "who you are" is to enter the whoami command:
* So now we need to understand where we are located, so we can easily navigate from that location to the one we need.

**What are directories?**

* **Pwd**

To Check what directory you’re currently in. The pwd command prints the name of the present/current working directory (PWD - Print Working Directory)

* **ls**

To Check what you have under that directory/folder. Lc command will list all folders and files you have inside that directory.

* **ls -la**

To Check what you have under that directory/folder including hidden files.

* **cd folder name**

Change directory: To further inside some folder.

* **cd ..**

Change directory: To go back into previous directory/folder

* **cd~**

The directory in which you find yourself when you first login is called your home directory.

* **mkdir**

Directories/Folders are created by the following command:

**Give Task: create on Desktop folder → go inside → create another folder → move to that→ go back to home directory**

Explain in detail that how it works,

When we open up the terminal and type in pwd then it prints a path, this location is actually where the terminal is by default , it is also called home directory.

Now to see the available paths in front type ls .this shows the list of directories inside the home directory.

Now type cd directory name to choose the folder where you want your terminal to go.

Use cd .. to move terminal back

And use cd ~ to move it back to home directory

--------------------------------------------------------------------------------------------------

* Now we know that why do we need to use GIT and GITHUB
* We learned how to navigate in different directories using Terminal

**Navigate to GitHub and show how the codes are maintained.**

Now,

* Lets create a GITHUB account and add some basic settings in our GIT

1. **First, create a GitHub account** (**please remember your email and username**)
2. So in this account you will be storing your code that you can share with your friends for now and later of course colleagues.
3. Now we need to introduce yourself to git and understand which code/folder we want to keep track of.

First navigate to Home Directory

**Since git is a tool, it has its own command that it can understand, all git commands will be starting with git …..**

* **git config --global user.name "name"**

Sets the name you want a ached to your commit transactions

* **$ git config --global user.email "email address"**

Sets the email you want a ached to your commit transactions

* **git config --list**

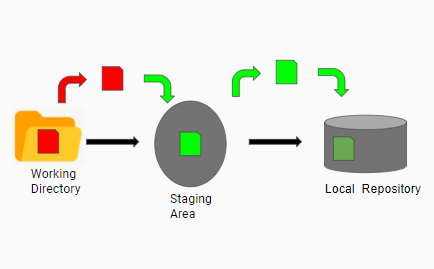
List your configurations

----------------------------------------------------------------------------------------------------

* We have set up our github account and configurations of GIT
* Before we jump into using GIT we will be looking into how GIT works

**SLIDE-17**

**Three Stage Architecture**



Git has a three stage architecture

* Working Directory:

Working directory is our current folder where all our files and folders are which we want to track using GIT.

* Staging area:

Staging area contains the files and folders we want to be in our next commit

* Local Repository:

Local repository contains all our commits with the message and unique code assigned to each commit.

Commit takes a snapshot of our code and saves it.

**SLIDE-18**

What's the advantage of this three stage architecture?

If we have 3 files in our working directory

1. Main.java
2. Car.java
3. Trucks.java

We made some changes to the file Car.java and Trucks.java

Trucks.java is giving some error and we do not want it to be in our next commit but

We do want to add Car.java update to our next commit

For this purpose we will add Car.java to the staging are and then make the commit.

----------------------------------------------------------------------------------------------------

We learned that how three stage architecture of GIT is helpful for us

**Tracking our First GIT Project**

* Create a project in intellij
* Navigate to the directory where the project is created
* Show how to navigate -->find path from the IDE and then navigate there

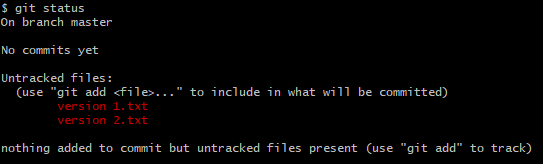
Now we want to track this project using GIT

We will be following these steps

1. Initialize a GIT repository using the command git init, it will show us the message “*Initialized empty Git repository in C:/Users/….*”

* Goto the folder see if there is hidden .git folder made

1. git status to check the current status of files that we are going to track



3.To start tracking our files lets first add them in staging area using the command git add . or git add “filename”

4. To commit the changes in the staging area we use the command

Git commit -m “type a message here”

Thats how we make our very first commit

**Play around making changes in the code use git status to view the changes and git add to stage , then add another commit**

5.Git log to view the commits that have already been made

6. Git ls-files to view files in the staging area

7.git rm --cached filename to remove file from staging area only

8.Git rm “filename” remove file from the staging area and directory

9. Git restore . or git restore “filename” moves file from staging area to your working directory *use this command if you made a mistake and want to go back to file that you have in staging area*

10. Git checkout -f move file from last commit from repo to your directory *use this command if you have staged the wrong file and want to restore from repo*

11.git restore --source commit code “filename” to restore a file from a particular commit *use this command in order to roll back to any of your previous commit*

12. git diff show difference between staged and working directory

13. Git diff --staged shows difference between staged and last commit

14.gitignore a text file containing all files which you don't want git to track

* Touch .git ignore -->creates a .gitignore file
* Add file name to gitignore which you don't want to track

------------------------------------------------------------------------------------

We learned how we can start to track our folder by initializing an empty git repository and use of different commands

Note:

Please make sure that you initialize the git repository only once in the directory which you want to track , most students make this mistake of initializing multiple repos within subdirectories and getting confused in between.

**Pushing Our code to GITHUB**

* Goto GITHUB.com login
* Create a new repository
* Type git remote -v command on your terminal to see if there is any remote attached to your current git repo
* If no then use command git remote add origin https:// to connect your current repository with remote github repo
* Git branch -M main create a branch main in the remote repository
* Git push -u origin main push the code in the branch main on the remote repo

Note:

**A common mistake**

* Your remote repo has different files and folder
* Your local repo has different file and folders
* Trying to push local to remote will cause an error
* Change your remote if you want to push to another repo

Use command remote remove <name>

Recap:

1.What is GIT and Github

2.What is three stage architecture.

3.How to run commands on local

**What are Different TYPES OF VCS**

1)**Local VCS**: **SLIDE-19**

A software in the local computer that keep track of all the versions. Everything is stored in your computer and nothing is uploaded on the internet.

* Pros:

Everything is maintained locally and there is no need for the internet.

* Cons:

If local machine wears out all the code is lost

**2)Centralized VCS**: **SLIDE-20**

John resolved this problem by making a website on which codes were saved on the internet.

Every person downloaded the code and did the changes required, after which they saved the updated code on the website again.

* Pros:

If local machine of someone fails they can still get access to codes saved on the website

* Cons:

If the website fails then no one will be able to get access to previous versions of codes, because they download the updated version of code or may be some previous version on their local system but the rest of the versions will be deleted.

**3)Distributed VCS**: **SLIDE-21**

In distributed VCS everything is the same just like centralized VCS, the only difference is that whenever some downloads the code it downloads the whole backup of code, so that we have access to all the changes that have been made so far.

* We learned about different types of version control systems

We have learned how to connect to GitHub create repo and push our codes to repo

**GIT Pull and GIT Fetch**

**GIT Fetch:**

**git fetch** is the command that tells your local **git** to retrieve the latest meta-data info from the original (yet doesn't do any file transferring. It's more like just checking to see if there are any changes available).

Example: show an example of git fetch where the remote repository is ahead of commits from the local repository and running the fetch command actually shows that there are some additions in the remote that local needs to be synchronized with.

**GIT pull:**

GIT pull is going to update your local repository to the state of your remote repository

Assume that there are two user working on the same GITHUB Repo,

USER 1 will use the same local repo we pushed on the git hub

USER 2 will clone the repo from GITHUB .

**Cloning a git repo and git pull to updated code**

* Create a directory on desktop named clonetest (assume it to be a USER2 on some other system)
* Use git clone url to clone the code of a repository already made on github by your account(assume that this is USER1 )
* Now USER1 push in your repository some updated commit
* USER 2 Goto your clonetest directory and pull the code and see it is updated or not

**Merge conflict:**

* Now USER 2 update the code in your clonetest folder and make a commit
* USER1 Make some updates in the repo that you pushed on GITHUB and make a commit and push the code to GITHUB
* USER 2 Now push the code in clonetest folder to github
* It will say that an error is there because your Repo isnt upto date
* USER 2 Use GIT PULL to update your repo of clone test folder
* IT will update the repo and give you and Error saying Merge Conflict , as there will be changes that you have made in the code in this repo also
* Resolve those conflicts by choosing which one of the code you want to keep
* USER 2 Use git push now and push the updated commit to the github from cloned repository

---------------------------------------------------------------------------------------------

We learned how to clone another repo and pull codes from it

We also learned that if the User 1 has updated the repo and User 2 try and push in the same repo without updating the repo it gives an error

We also saw that if there are some changes made by the User 2 and if he pulls the code it gives him a MERGE conflict .

HEAD—remote and local synchronization----branch name

**Stashing**

Why do we need stashing?

Lets assume you are working on a project

* While working you get an amazing idea that can make your code very efficient and clean, but you are not sure whether it will work or not.
* But you want to try it out first and it will take you some time to implement that idea.
* You decide to work on project and in parallel when ever you have free time you want to work on that amazing idea.

.

How will you save the progress of your idea?

You can not commit it as you aren't sure it will even be used or not?

**How does stashing works?**

* stashing basically stores the snapshot of the current file in a separate container revert the original files to last commit
* each step can be stashed and the original file can be taken to the desired stashed step and committed if needed
* **It will revert your changes in staging area also to last commit**

Commands:

* git STASH

Make a separate copy of code and revert the one in directory to the last commit

* Git stash -u

Stash untracked files

* Stash list

List all stashes

* Git stash push -m”my first stash”

Push the stash with a message to identify

* Git stash drop “index number”

Drops the stash i.e deletes it

* Git stash pop “index number”

Delete the stash and restores it into your directory

Now you can commit the final changes in your code and push it to github

**EXAMPLE:**

*Practically demonstrate it on an example of some code*

*Build three cars*

*USE CASE 1:*

*Commit after making classes*

* *Add features for car 1* 🡪*stash*
* *Add features for car 2* 🡪*stash*
* *Add features for car 3* 🡪*stash*
* *Visualize all 3 cars by changing stashes and re-stashing them*
* *Show that if we are having a stash applied and try to apply another stash over it gives us an abort error(so we are left with two possibilities 1. To commit the change or 2.stash it first and then apply stash)*
  + *First stash it and show by applying another stash*
  + *Then commit the stash and create a new stash (it will say no local changes to commit)*
  + *Now apply any previous stash it will give us a merge conflict.*
    - *Why do we get a merge conflict here ?*
    - *Because when ever we stash git just stashes the changes and reverts code to last form in repo , but now the code in repo has been changed and now when we reapply any previous stash and the lines on which there was no code at first ,it finds a code it will give us a merge conflict. Makes sense right? Because it doesn’t know which one to keep.*
  + *Once we have such a scenario and we merge it , then we can not got back to our previous changes because we need to commit it now or add it to staging are , only then we will be able apply another stash.(why is this behavior been seen?)*
* Now reapply changes and take the code back to initial stages where the car was not made commit it and show that there is not a merge conflict
* Now commit the changes and choose a car , now make a new file and show how we can stash another file if its untacked and if its tracked then how

USE CASE 2:

* Make a commit
* And started to add more features in the file
* Figured out something needs immediate change in the commit
* Stashing the changes and making another commit asap
* Now you can reapply the stash and get all your work done back
* Show that if you do not stash it gives an error asking you to stash them or commit them

USE CASE 3:

* Make a commit
* Upload it to github
* Now make some changes on github file
* Make some changes on local file
* Now do a gitpull after stashing your changes to see the updates
* Now do git pull without stashing and show a merge conflict
* Show how to resolve this merge conflict

*Note:*

*If you have committed the code and now you apply a previous stash it will give you a merge conflict*

---------------------------------------------------------------------------------------------

COLLAB ON GITHUB

Suppose you want your friend to work with you on the same project

For this purpose

Follow these steps

1.goto your repo you want to work on with a friend

2.Click on manage access

3.In invite collaborator enter user name and send invite

4. Your friend will accept the invite and you both have access to this same repo and can commit your codes here

**COLLABORATIVE TASK ON GITHUB USING STASHING**

**Make a group of 5 people and choose one leader**

**Leader will create a repo and send an invitation link to their group fellow**

**Accept the request**

**Group Leader tasks**

* **Now the leader of project will create a project**
* **The project should contain two files**
  + **Main .java**
  + **Car.java**
  + **The files to be uploaded in the repository are Src and .gitignore**

**In Main.java write the following code**

**package com.company;**

**public class Main {**

**public static void main(String[] args) {**

**Car Mercedes= new Car();**

**Mercedes.color="Black";**

**Mercedes.number="4000";**

**Mercedes.model="QAX787";**

**Mercedes.engineCapacity="1500cc";**

**Mercedes.seatingCapacity="2";**

**Mercedes.name="Mecar";**

**}**

**}**

**In Car.java write the following code**

**package com.company;**

**public class Car {**

**String name;**

**String number;**

**String model;**

**String engineCapacity;**

**String seatingCapacity;**

**String color;**

**}**

**Group members tasks**

**Clone the repository in your system**

**Make any change to you main.java file and commit it**

**Push it on github**

**Review the problems faced and try to resolve them**

**Problems:**

* **The first person to commit will be able to commit sucessfully**
* **All other persons will face an error saying that (“**Updates were rejected”)
* **Stash the changes you made and use git pull to update the repo and then apply stash , resolve merge conflict (use the updates you made and Delete the other ones)**
* **Try to commit again**
* **Incase it fails again (because the next person would have commited his changes)**
* **Apply stash again use pull to update repo again**
* **Try till you are able to sucessfully update the repo**

**-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**We learned**

1. Why use git and git hub
2. How to navigate using terminal into folders
3. Creating a github account and adding username and email in git
4. Tracking our first git project and exploring different useful git commands
5. Pushing our code to github
6. Cloning a project from github and pulling code over the period of time to get updated versions
7. Why do we need stashing
8. How to use stashing effectively while working on a project
9. How to collaborate on GITHUB

**Branching in GIT**

**Why do we need Branching?**

**CASE 1: ADDING A NEW FEATURE**

Suppose three friends Tony, Mike, john are working on a project

The project is working fine and all three of them are working in collaboration.

Meanwhile Tony says that I have an amazing idea to add a feature should I add it?

Mike and John although they trust tony but their project is already about to end so they ask Tony to create a branch and work on it.

So tony can create a branch on which he started working in parallel to the code in that branch.

John and mike continued to work on their main project

So the following scenario will happen

* Tony will be working on a new feature in a new branch
* Mike and john will be working on the master branch in the same project

Once Tony is done with his work he will tell mike and john, they will look at his feature if they approve of the changed he has made

They will merge the addition that Tony has made into their master branch on which they both have been working

**Case 2 A practical scenario🡪 Testing the functionality of an AUDII CAR**

Lets assume that you as being a tester have to write a code to test the functionality of an AUDII car

The Test must cater the major test scenarios and there is no limit to the minor ones

Mike , John and tony all are working and the are testing the following features

* Air pressure
* Headlights
* Max Speed
* Vibrations
* Fuel Bar

They are the major features that need to be tested

Now John thought how about adding some new test cases

The **problem** john is having write now

* He cannot leave the main test cases as they are necessary
* He isn’t sure whether those test cases will be approved or not
* He cannot add them in the Branch he is working on
* He can still use stashing but every time he has made a commit and will reapply the stash it will give him a merge conflict which he doesn’t want

The **solution**

* John can work on those test cases in an entirely new branch and his work in separate branch will remain isolated from his original work
* Once he is done with the them he can ask for a review and merge them

**Other use cases of branches:**

Branching has a large number of use cases that help us work efficiently whether our projected is in development phase or in the production and updates are been given

Suppose that you made an automation testing script and it is deployed now

* Resolving an error in the script 🡪 Suppose a bug was found in the code deployed, you will not change the master branch as the original code Is there and you do not want to disturb it. So you

Will create a branch in parallel resolve the error and then merge it in the master branch

* Incorporating new features
* Testing any new idea
* E.t.c

**Git Branching Workflow in Production:**

**Long Running branches:** are the branches that are there for a long period of time

The example of them are Master , Develop and Proposed update branch

In Master Branch there is our main code of production

In Develop branch we have the development and testing, after the tests are cleared we merge them to master branch

In Proposed update branch the new updates are proposed, if it is agreed upon it is merged into develop branch

Three developers are working on them and are pushing and pulling in the branches, and one of them is the lead who takes control of merging.

**Topic Branches:**

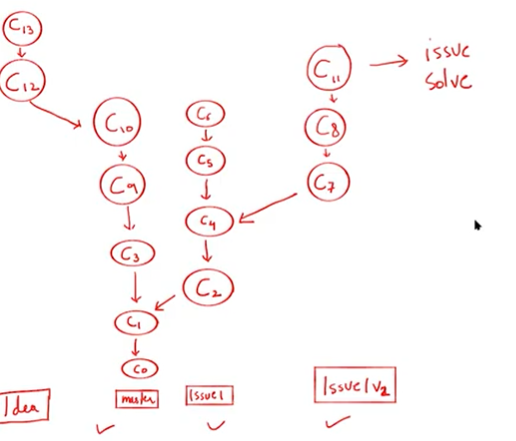
Lets say that one of the coworkers say that lets use a plugin e.g typejs and lets replace the previous plugin as this new one is more effective

So he will create a new topic branch with name **replace the plugin** and once it is tested and this plugin is working fine, testing has been done, then we will merge this branch to master and delete this topic branch

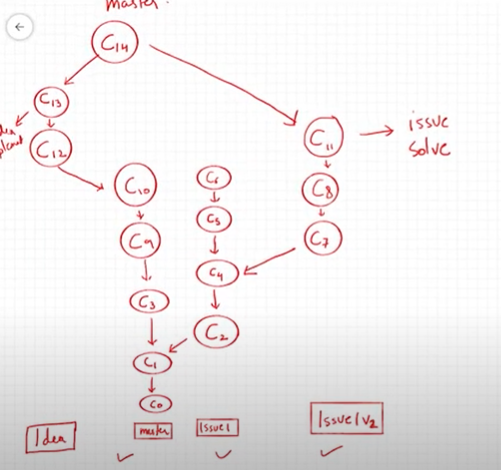
Topic branch are short lived.

Example :

You make a commit in master branch and there is an issue , you can create the branch issue 1 , if there is another issue you make another branch issue 2, C12 and c13 are new commits of new idea



Now we have resolved the issue 2 and we have completed the new idea so we will merge them to master making a new commit c14



A good practice is that master branch is left for the deployed code and all other work of development and testing is done in separate branches to avoid any problem in the main original code.

Outcome:

Now we know that why do we need branches and a basic workflow of branching

Example:

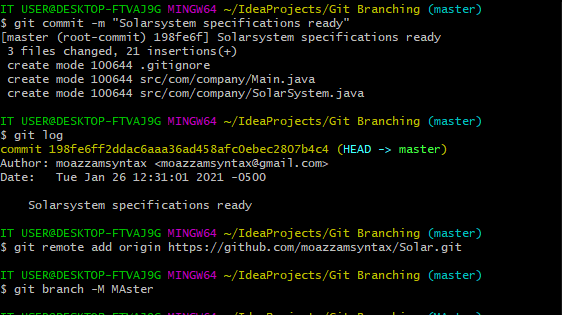
Now lets implement an example and see in detail how we make use of branching

* Open intellij and create a new project

1. **In a class Solar system**

package com.company;  
  
public class SolarSystem {  
  
 int Sun;  
 int Moon;  
 int PLanets;  
  
}

**make a commit, name it solar system specifications ready and push it to your remote git hub repository in the main branch**

****

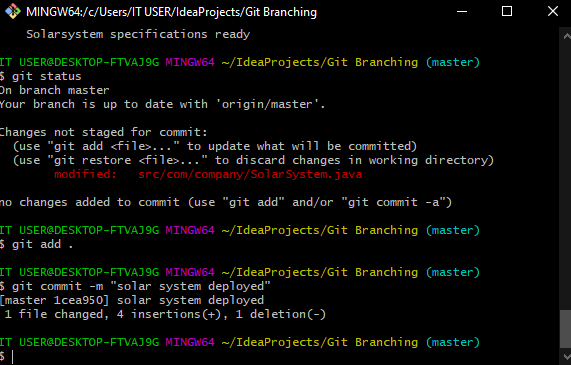
Now we will discuss some important concepts over here regarding the commands

**The git branch -M master** is used to name our branch and its our choice to name it as we want, as we currently have one branch lets name it master, we can also name it any other name if we do not want it to be the master branch

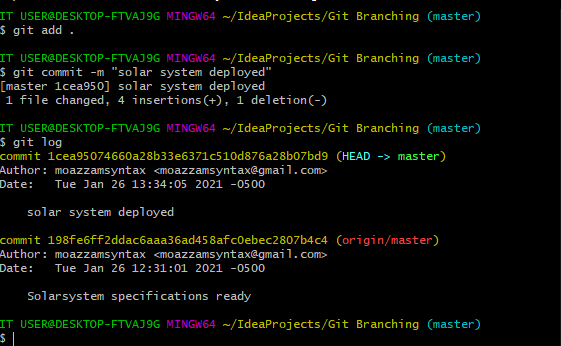
**Git push Origin -u master** we are telling the git to push our code to the repo **ORIGIN** in the branch name **master,** we can change the name of the repo also From origin to any other while adding remote

1. **In class MAIN**

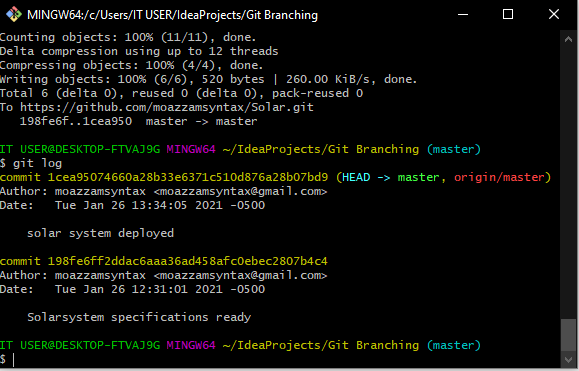
package com.company;  
  
public class Main {  
  
 public static void main(String[] args) {  
   
 SolarSystem FactorX =new SolarSystem();  
 FactorX.Sun=1;  
 FactorX.Moon=3;  
 FactorX.PLanets=2;  
 }  
}



* Now we have deployed our solr system and its in a very basic form , lets say that we want to earn right away and clients said that deploy intial version asap and continue working on features
* Make a second commit and use **git log** to see the commits



* We see that the using git log command our head is pointing to the latest commit we made and **origin/master** is on the previous commit , because we haven’t pushed it yet to git hub
* Use Git push and push this to github and we will see **origin/master** also updates



* We want to work on some more feature now should we work on the master branch or create another branch?
* Obviously if we started to work on master branch and made some mistake our deployed code will be effected so what we do is that we create and other new branch

1. **Create a branch Adding Feature to sun:**

* git branch FeaturesSUN-🡪 create the branch
* git checkout FeaturesSUN 🡪switch to the branch
* Add a new class named sun in here and add this code

public class SUN {  
   
 string Heat;  
string Radius;  
int Temperature;  
   
}

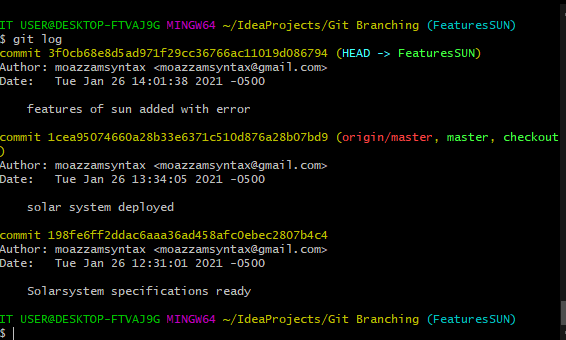
* In main add the following code

SUN Freon= new SUN();  
Freon.Heat="500kj";  
Freon.Radius="7000km";  
Freon.Temperature="40000c";

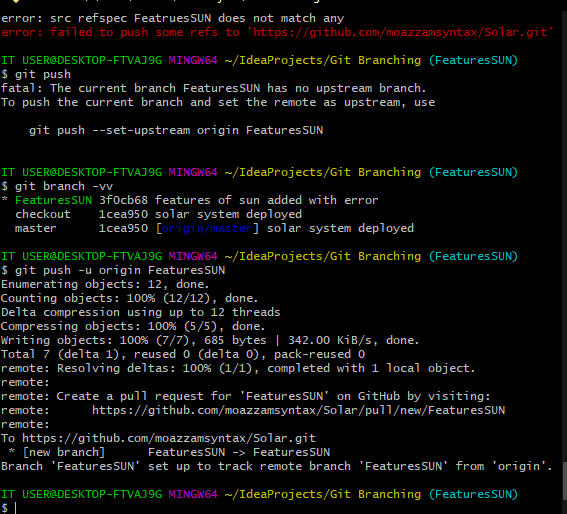
* Now lets assume that this code wasn’t that simple and had hundred of lines and we made some error just like we did , assigning strings to int and we are not able to figure this out and we also donot want to

Make more changes in this code because it may disturb anything else and rest of features are working perfectly fine

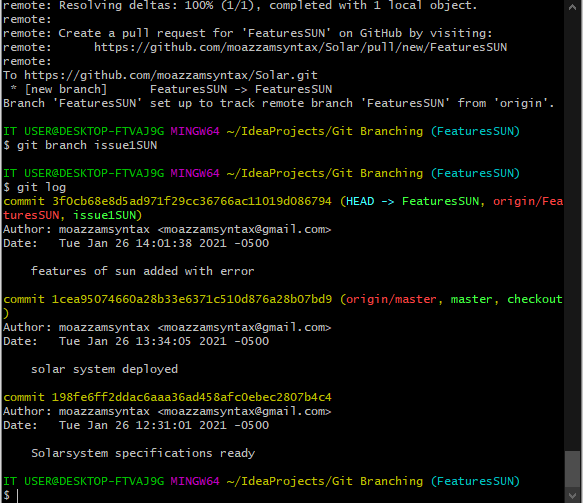
* Make a commit on this branch



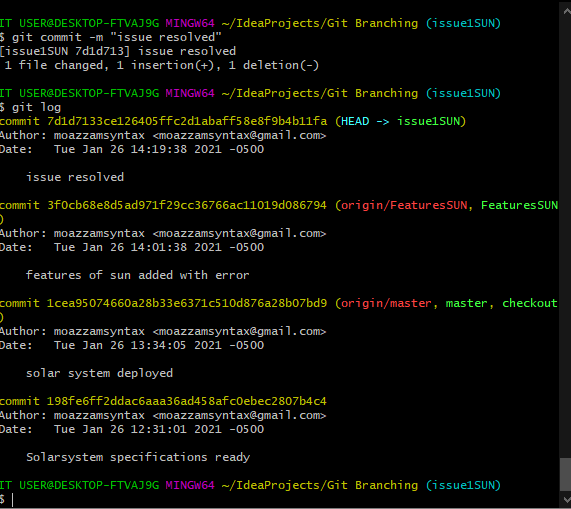
* Push it to github



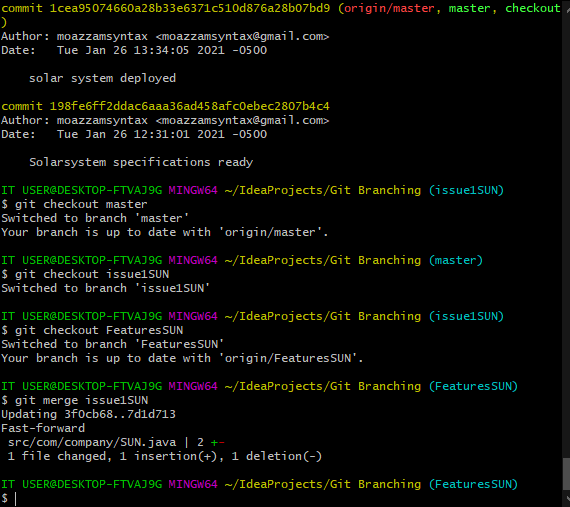
* + Use git branch -vv to see the current status of branch
  + And git push -u origin FeaturesSUN(or SUN)to specify in which branch of origin the push has to be made



* + Using git log We can see here that both the origin of branches are up to date with remote now
* Create a new branch name issue1SUN (or issueSUN)
* Make the updates in code resolving the error
  + public class SUN {  
      
     String Heat;  
     String Radius;  
     String Temperature;  
      
    }
* Make a commit with name issue resolved (commit is necesasry because if you wont commit it you cant change the branch git takes care of that).

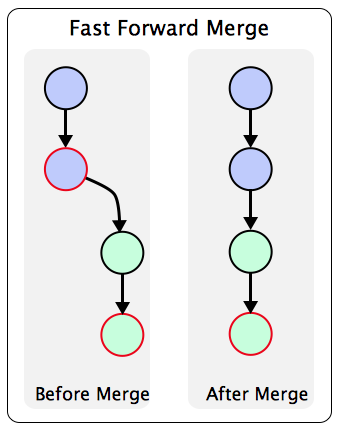


* Checkout all 3 branches 1 by 1 and show all the code is intact in all branches .
* Now we have resolved our issue and wants to merge our branch to the Previous branch SUN
* Checkout to the branch to which we want to merge and use command git merge issue1SUN



* We will see that Fast Forward Merge Happened and the issue will be resolved in our branch FeaturesSUN
* Now if we do a git status it says no changes to be commited . WHY?
  + Because the merge made a new commit automatically
  + Do a git push and goto github and see the commit updated in branch FeaturesSUN and with the name issue resolved

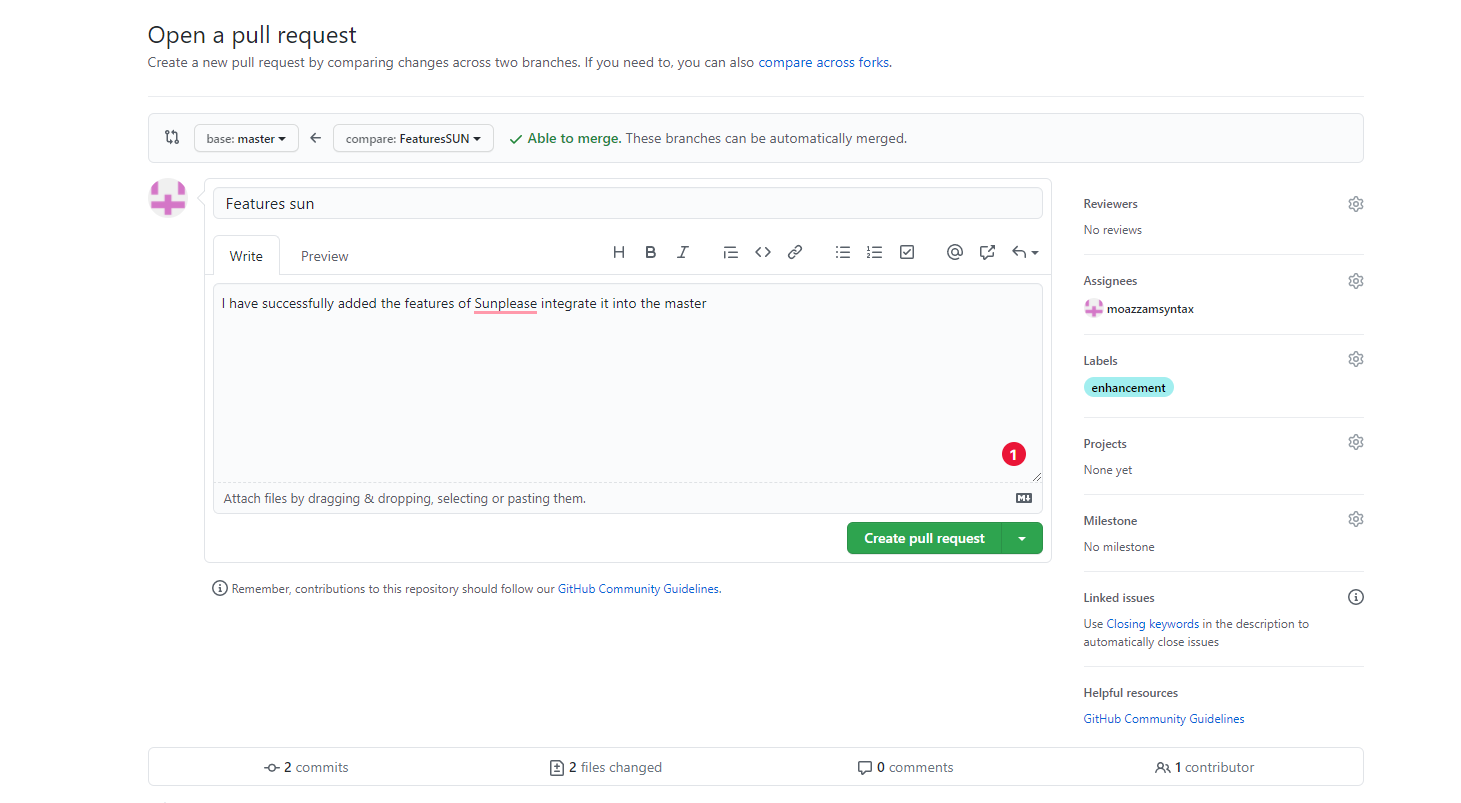
**Here we need to understand that when does a Fast Forward merge happens and how it works.**

****

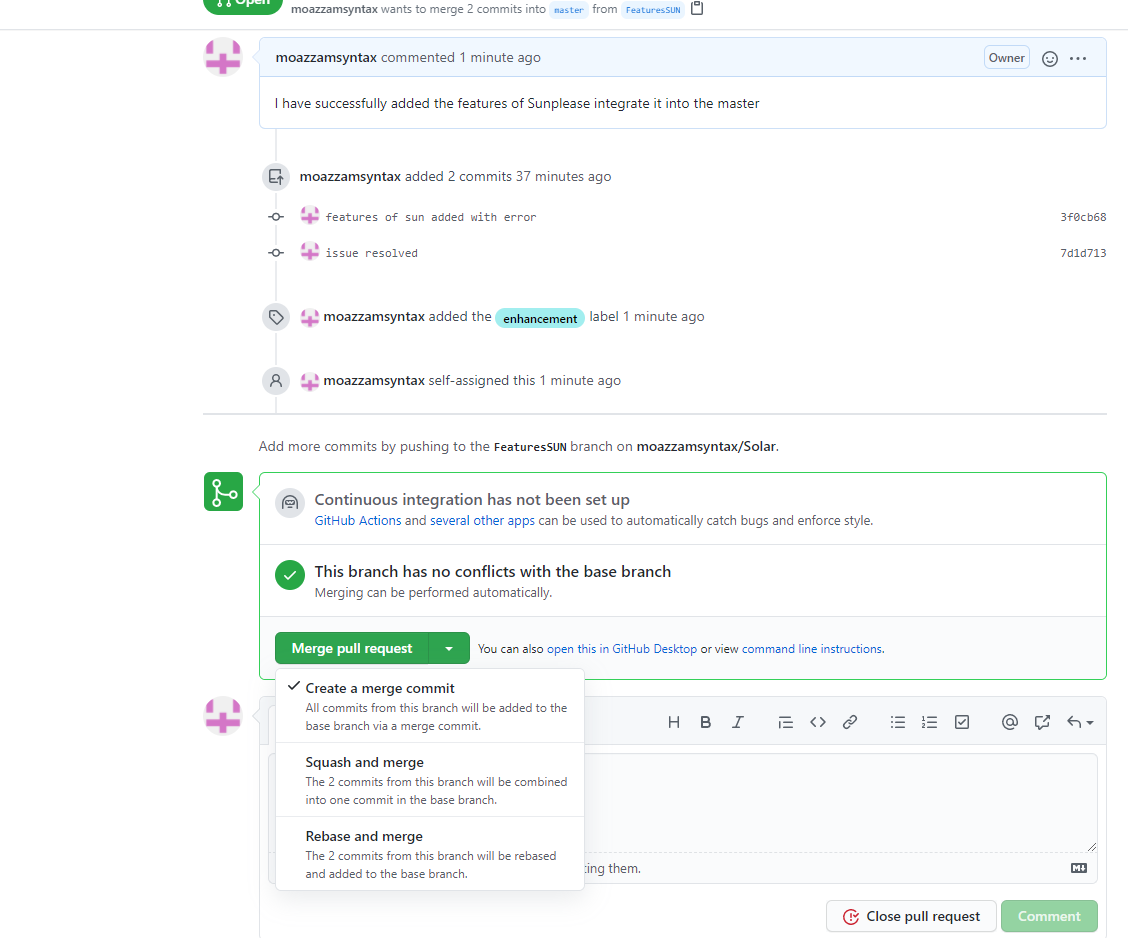
* Git uses a **Fast forward merge** technique when there **is no other commit** in time duration when a **new branch** was **made** , **worked on** and **merged**
* Infact the merge is simply that it starts pointing to that latest commit instead of actually merging them together cause git knows this is the final version we want to keep

**Lets merge our branch Feature sun to the master branch as it has been completed now**

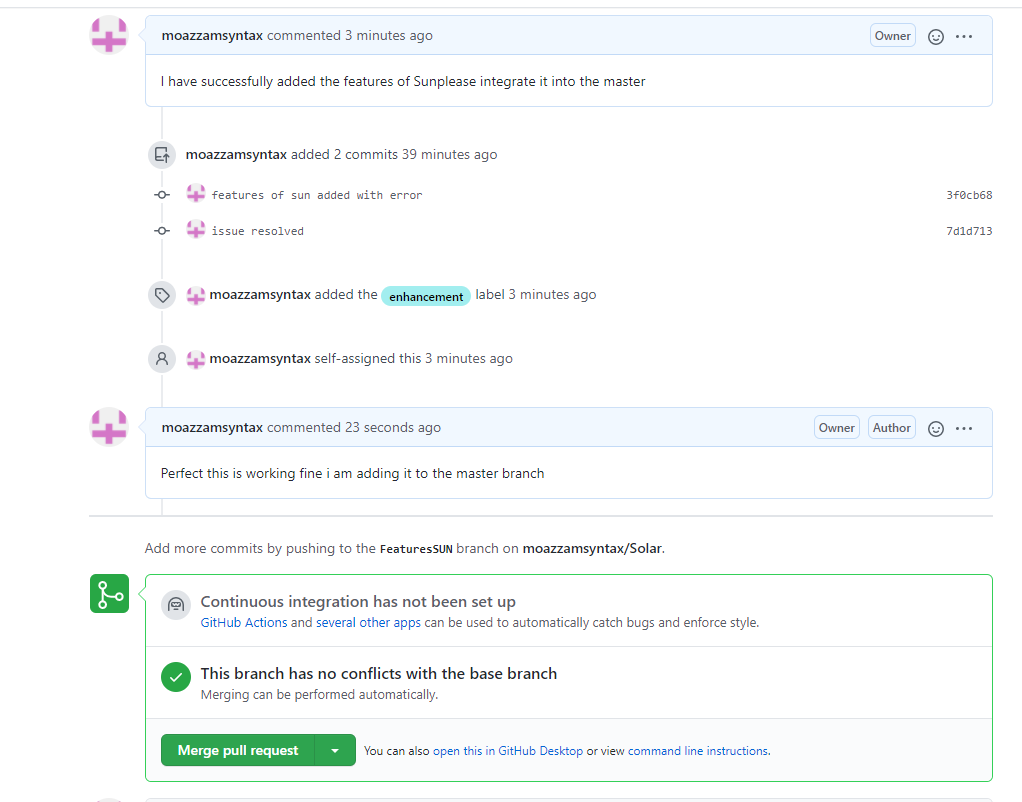
* Goto GIHUB create a pull request



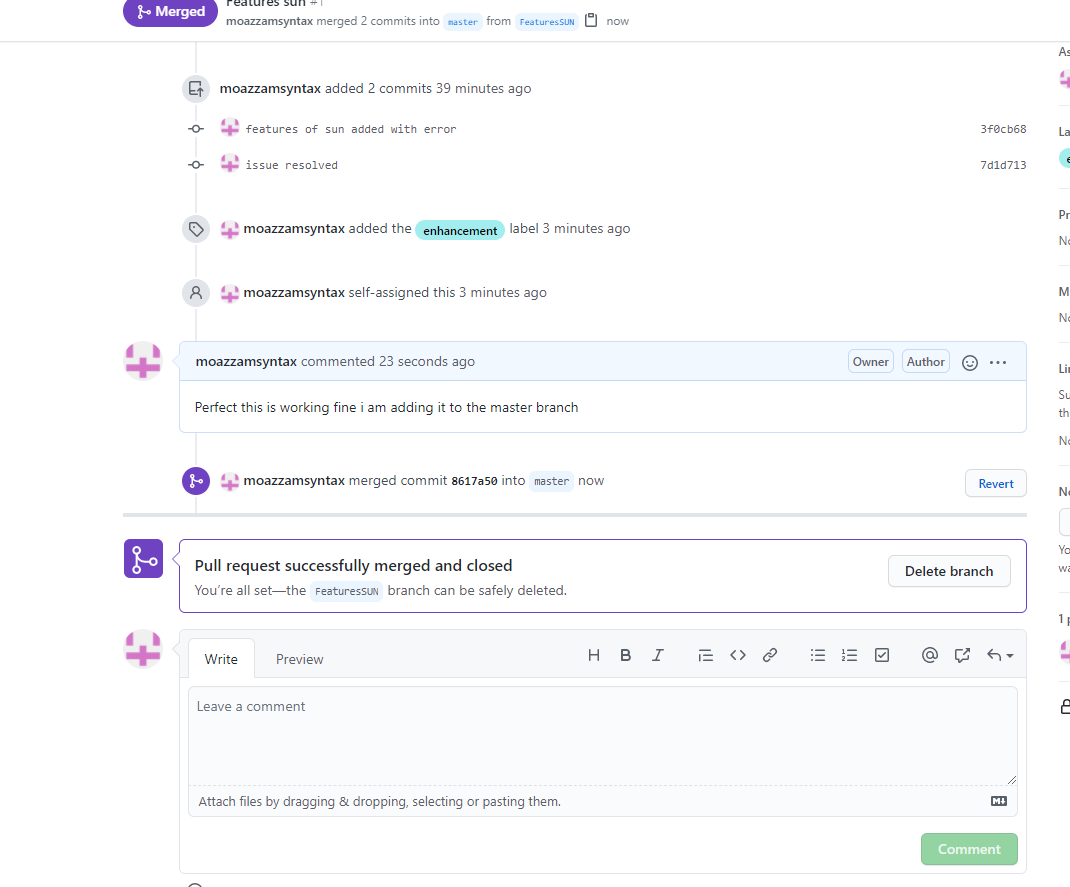
* Select the type of Merge you want to make
  + Git **rebase** and **merge** both integrate changes from one branch into another. Where they differ is how it's done. Git **rebase** moves a feature branch into a master. Git **merge** adds a new commit, preserving the history



* Add a comment if any issues let the person know that why this cant be yet done

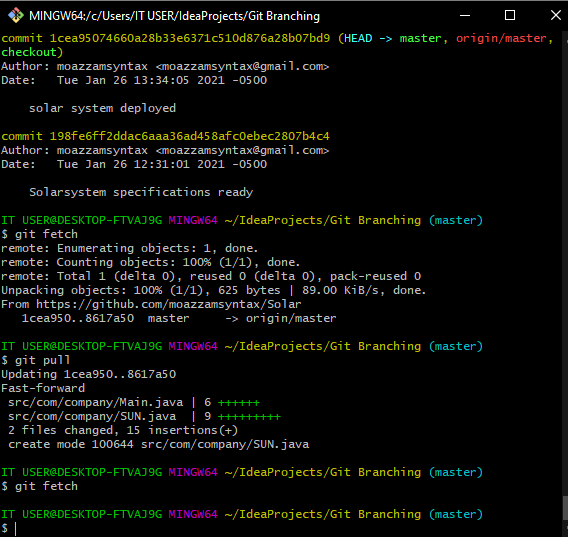


* Merge and Delete if you want to or keep it as it is over there

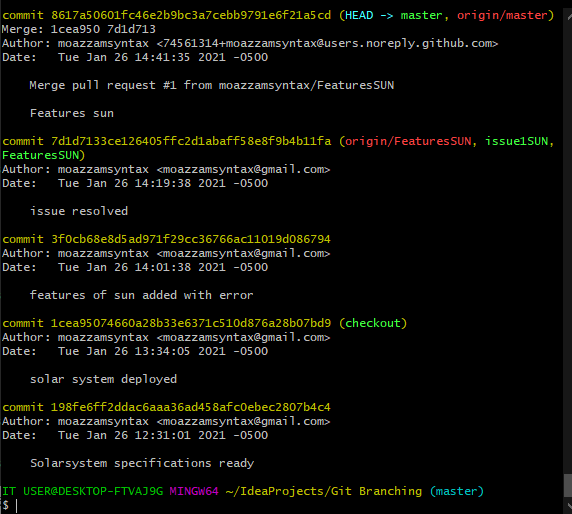


Now lets update our local repository so that the merge we did on GITHUB is also done in our local repository

* Use git fetch command



* + We see in the fetch that there is one update that needs to be made
  + And the update is in master
  + Now use git pull to update the complete repository with GITHUB
  + Use git status to see the commits that have been made yet



* + We see that our commits are now total in number 5 because the last commit we made on GITHUB is added as a new commit on top of all other commits

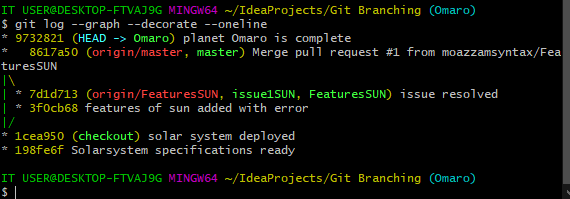
1. **Now Lets assume that two friends are working in parallel on the same project**

The next feature that are to be added in our project are the specifications of the planets

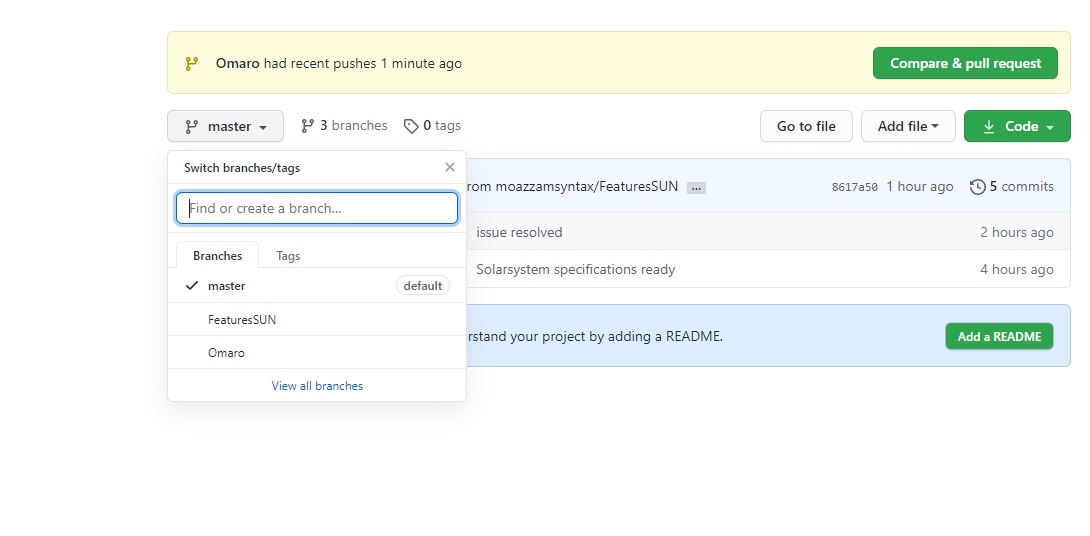
* Create a desktop folder with name “User2” and clone the whole repository in this folder

**USER1:**

* Create a new branch named Omaro.This is where user 1 will add updates for his planet.
* Git branch Omaro
* Git checkout Omaro
* After switching to the branch add a new java file named planetOmaro
* Add the following code in class
* public class planetOmaro {  
   String Size;  
   String color;  
     
  }
* Add the following code in MIAN
* planetOmaro Omaro =new planetOmaro();  
  Omaro.color="Red";  
  Omaro.Size="100000km";
* Commit the code on this branch
* Use git log git log --graph --decorate --oneline to see the structure of branches yet made



* Push this code to GITHUB using
  + - git push -u origin Omaro
* We have a total of 3 branches in our GITHUB now



**USER2:**

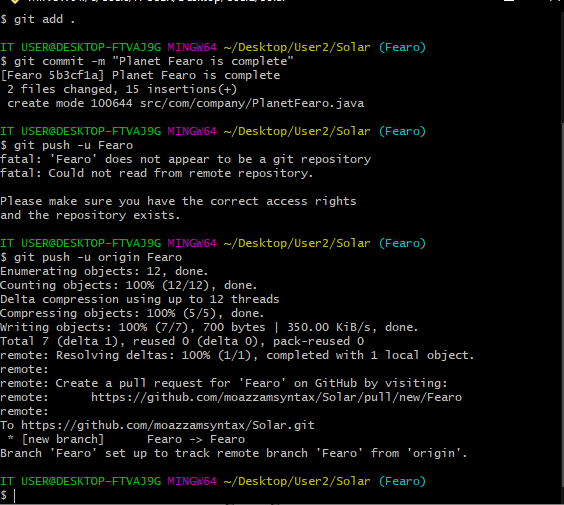
* Create a new branch named Fearo.This is where user 2 will add updates for his planet.
* Git branch Fearo
* Git checkout Fearo
* After switching to the branch add a new java file named planetFearo
* Add the following code in class
* public class planetFearo {  
   String Size;  
   String color;  
     
  }
* Add the following code in MIAN

planetFearo Fearo =new planetFearo();

Fearo.color="Blue";

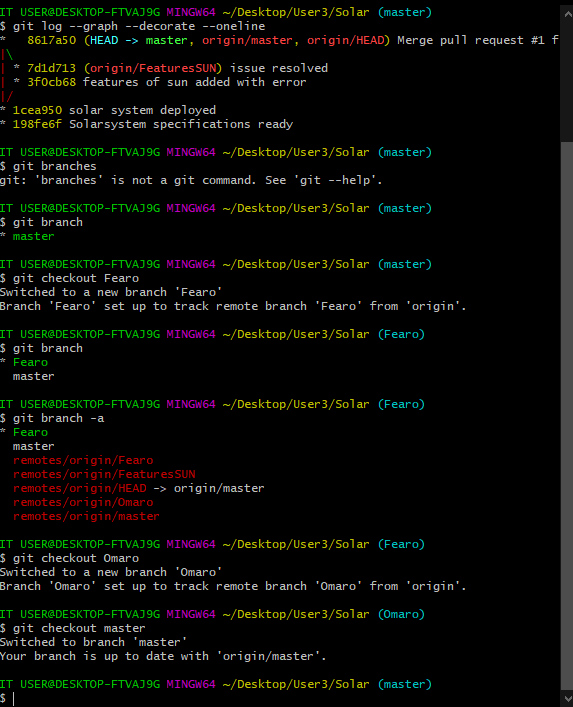
Dearo.Size="90900000km";

* Commit the code on this branch
* Now push it to github, we wont need to make a pull request as we are working on a seaparate branch and hence we saved a lot of hustle from getting into conflicts



**User 3: testing the updated Features and combining them into one file**

* Create a new desktop folder named USER3 and clone the project in that folder
* By default, git clone creates only one branch: the currently checked out one, generally master. However, it does create remote tracking branches for all other branches in the remote. Think of these as local copies of the remote's branches, which can be updated by fetching. They're not real local branches, as they're intended only as pointers to where the remote's branches are, not for you to work on.
* In order to see all the branches use the command git branch -a (it will show you the remote branches)



* Switch t0 the branch you want to by using git chechout <branchname>

Ok so now user3 is responsible for merging this

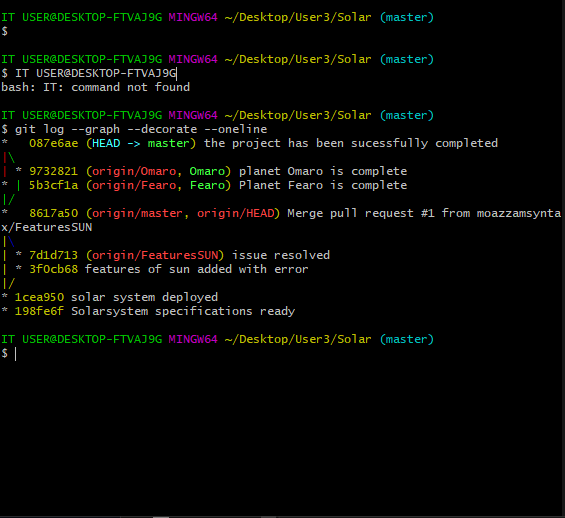
* He can test the code individually and if its ok to merge he can
* Lets say he wants to merge them now
  + Standing on master branch use Git merge
  + It will do a fast forward merge as there was no previous commit in the history in master branch
  + Now use git merge Omaro

Auto-merging src/com/company/Main.java

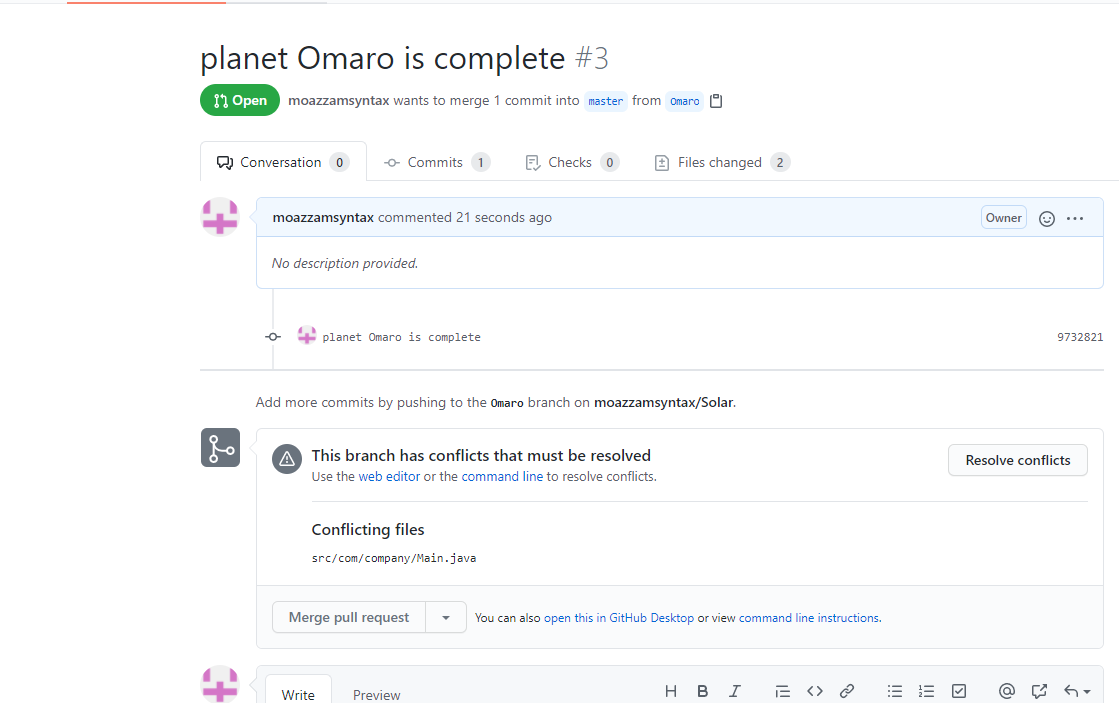
CONFLICT (content): Merge conflict in src/com/company/Main.java

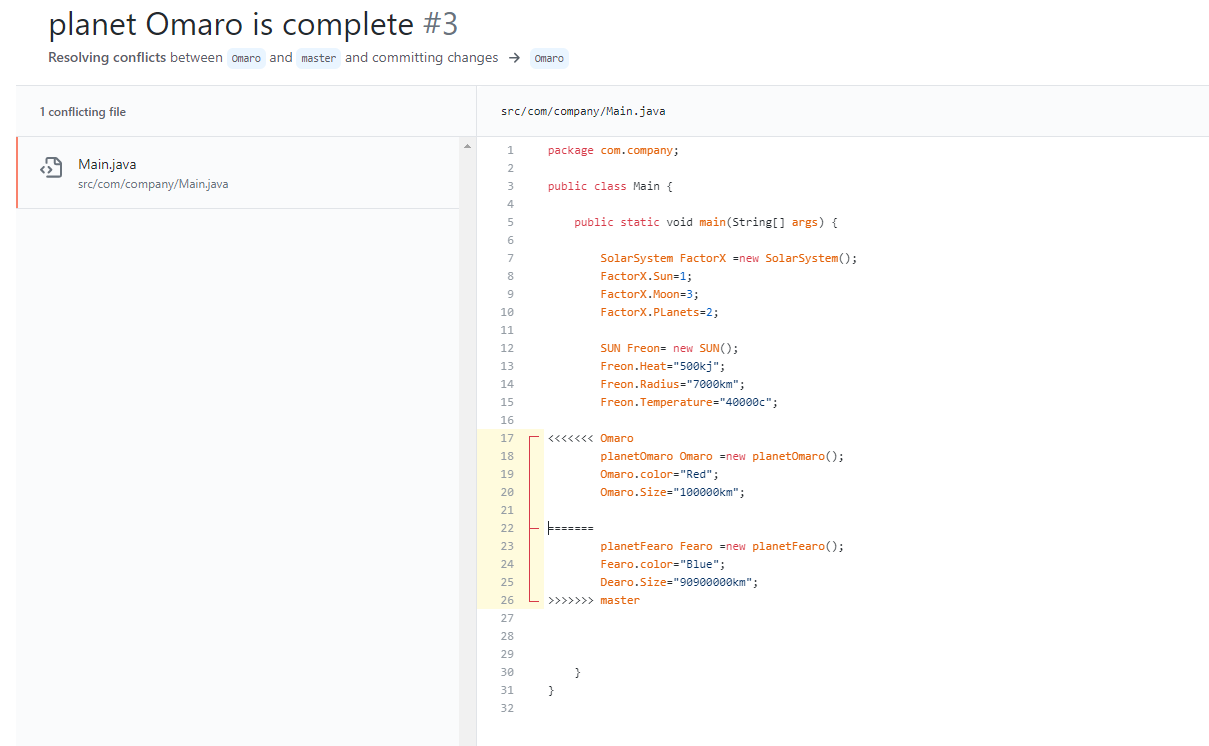
Automatic merge failed; fix conflicts and then commit the result.

* + We see this conflict because now git can not decide that what to do with the conflict faced as the previous commit history has been updated from when the branch Omaro was created
  + To resolve the confflict open the file in which there is conflict and choose what to delete and what to keep
  + As we want all the data so we will just say that okay keep all changes
  + Use git log --graph --decorate --oneline command to see the branches



* Do not push the code to github and now we will do the same merge on github just like we did the previous one





git merge --no-ff <branch> to stop a fast forward merge and it will doa recursive merge which will need you to make another new commit

**The END**

