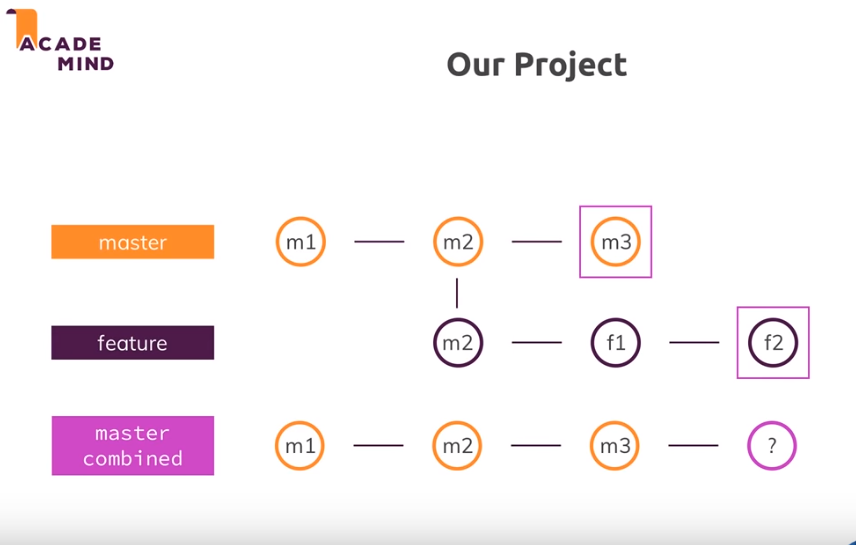
Here we want to have a look at git merge and git rebase and see how these commands work and how we can use them. The problem is well these commands can be applied in lot of different situations and have pro and cons and so on. I want to give you a quick example how you can use both of these. For this we will have a look at our example project in intro and then will continue with actual code, because in this video we will start with following situation-

we started with master branch, there we had 2 commit. then we created feature branch(from code of m2 commit), then we have commit f1 in feature1. In same time someone made another commit m3 on master. So now both branches have evolved. Then we continued to work on feature and made commit f2 in feature branch. Now question is how we can combine these developments m3(latest commit in master) and f2(latest commit in feature) into or back into our master branch right here(third image). For this we have lot of different approaches and in this video, I want to show you 2 ways to how to achieve this. One is with **git merge** and another with **git rebase** . this was theory, lets see some code now.

So in code we have commit s with name m1,m2,m3, in master branch and commits with name m1,m2 ,f1 and f2 in feature branch. So m2 is last common commit between 2 brnches.Now if you switch to master branch and run-

**git merge faeture1**

dnt run this command(git merge feature1). This wnt be a problem because this will merge branches and will create a commit in master branch which combines these 2 branches. But this will also merge all the other commits of feature branch into master branch. If you now run git log in master branch-

**commit 723b94af3f4cfa0ce3ab413ed2945a22e86cd5d5 (HEAD -> master)**

**Merge: d707b4e d27a8ad**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Wed Oct 17 22:55:11 2018 +0530**

**Merge branch 'feature1'**

**commit d27a8adbf0509a511fc1c1c2bcb7098120035183 (feature1)**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Wed Oct 17 22:54:06 2018 +0530**

**f2**

**commit d707b4e04168d77e55db2a8c7bda23eaf2ec9b47**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Wed Oct 17 22:53:07 2018 +0530**

**m3**

**commit 383c01155f478ee61a3517e0e5c63721b0a0099a**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Wed Oct 17 22:51:48 2018 +0530**

**f1**

**commit 3933fddd0593e1c3740f5a19199800f43ea4720e**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Wed Oct 17 22:50:40 2018 +0530**

**m2**

**commit c15271281506cee75956813886a9e53b75ae6344**

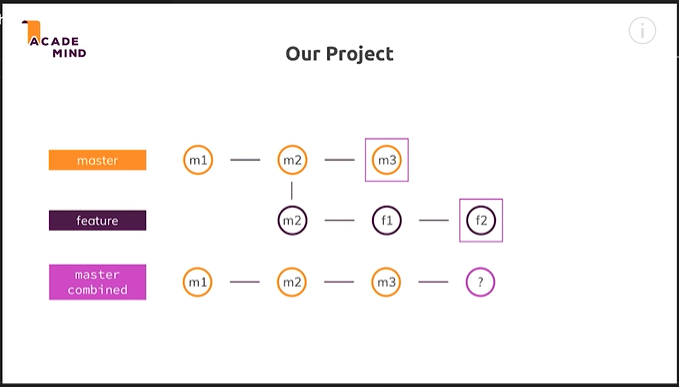
**Author: unknown <sumitsood3127@gmail.com>**

**Date: Wed Oct 17 22:50:15 2018 +0530**

**m1**

This is not a problem,this is quite nice actually because git is a tool that allow us to track how our project evolves, so what different commits we have, what different feature we implemented, which features we corrected afterwards and so on. If we want this, just run above command.

But I want something different ,I want this-



There we have 3 commits in master and after that I want to have kind of updated information of the feature branch because of that I will not use **git merge,** so I will use –

**git merge --squash feature**

**Automatic merge went well; stopped before committing as requested**

**Squash commit -- not updating HEAD**

squash kind of allow us to summarize all the different commits , so all the changes we had in feature branch in last commit(it kind of puts them togather) and then merges this last commit with latest commit in the master branch.

But as we can see in cmd, **Automatic merge went well; stopped before committing as requested.** This sounds like a problem but actual solution is , run –

**git commit –m”feature and master merged”**

here we gave the name of this new merge commit.so this was not problem, it just gave us change to name our new commit. now if you run git log-

**D:\gitTest>git log**

**commit 85f3ad5381ab67c757b3a4bcf58c93cad8c94337 (HEAD -> master)**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 06:54:47 2018 +0530**

**master and feature merged**

**commit 072979b754327b9fd317ba8861168f7706842159**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Wed Oct 17 23:09:28 2018 +0530**

**m3**

**commit e226c81235cdd83dee7128cb33d7704ae677ee23**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Wed Oct 17 23:07:53 2018 +0530**

**m2**

**commit 29b297a4cd3618c3ea548e5edc005087bf084b65**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Wed Oct 17 23:07:29 2018 +0530**

**m1**

so you can see that we have commits only of master branch – m1,m2, m3 and now this last commit. this is nice but you could say that will I do not like this because would’nt it be nicer if I would have structure (of commits) i.e, m1,m2,m3 then changes of feature branch but if I would see these 2 chnages right here f1 and f2, we also have singe commits in our feature branch, would be this right here(last merge commit, in master branch), which would allow me to better track the changes I made and so on. There is a way to do that and for this we can use **git rebase.** Now its time to rewind the project a bit.

So in master branch we have commits upto m3 but in feature branch we have commits upto f1 and feature branch is based of m2 commit.

now I want to use rebase command and it works a bit differently than merge. This is structure in feature branch-

**D:\gitTest>git log**

**commit 2298b1159304af506832b3140dedcf34e7358e47 (HEAD -> feature)**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:25:33 2018 +0530**

**f1**

**commit 84ddfe74400d64f61216f5f200b877f0436b9d18**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:24:13 2018 +0530**

**m2**

**commit 1d527294c13721de544516ec9a248d4d96d5d419**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:23:47 2018 +0530**

**m1**

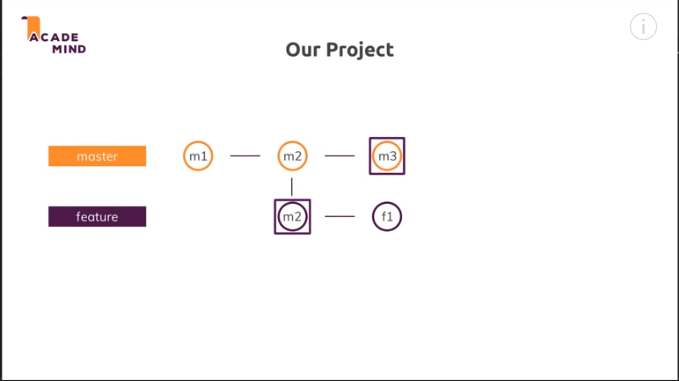
in feature branch run-

**D:\gitTest>git rebase master**

**First, rewinding head to replay your work on top of it...**

**Applying: f1**

master is another branch which has additional commit. now this 3rd commit in master branch should be the base of our feature banch.



So something happened. So if we run –

**D:\gitTest>git log**

**commit 52407674d3c99e87564aa00d52569816da15c213 (HEAD -> feature)**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:25:33 2018 +0530**

**f1**

**commit 531ee3b0e309b30bb81936b8a96c0ede28b9e2c1 (master)**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:26:33 2018 +0530**

**m3**

**commit 84ddfe74400d64f61216f5f200b877f0436b9d18**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:24:13 2018 +0530**

**m2**

**commit 1d527294c13721de544516ec9a248d4d96d5d419**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:23:47 2018 +0530**

**m1**

we can see that commit m3 magically appears in feature branch. If you go to master branch , you can see m1,m2 and m3 commits. So nothing changed there. Lets see what happened.

We were in feature branch and we ran  **git rebase master**, so first thing git does is, git checks both branches and has a look at last commit both branches have in common, which was that commit? it was m2 commit, commit we used to create feature branch initially. Then git has a look at our current branch, remember we are in feature branch. So git looks at this feature branch and see what changed actually in this feature branch, now I think it finds all the changes and then saves these changes internally for the moment. So nothing is committed or something like that. It’s just saved internally. Then git goes back to master and see what happened to master., so how the master evolved and what does git sdee riht here, it sees that master git additional commit the m3 commit. now git uses this m3 commit and also kind of moves the feature branch forward to this m3 commit. so the feature branch is no longer based on m2 commit but on m3 commit and hen both branches are aligned again and then git simply applies this internally saved information , so this f1 change and puts it on top of m3 commit. that is everything which is happening and why that’s cool in our case? Because now I can add f2 in feature/index.html, lets our feature is finished. The commit this change. Now if we run –

**git log**

**commit 5b4bd3c50b4ee8aab788dca17b84071f42c6b4b4 (HEAD -> feature)**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:51:41 2018 +0530**

**f2**

**commit 52407674d3c99e87564aa00d52569816da15c213**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:25:33 2018 +0530**

**f1**

**commit 531ee3b0e309b30bb81936b8a96c0ede28b9e2c1 (master)**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:26:33 2018 +0530**

**m3**

**commit 84ddfe74400d64f61216f5f200b877f0436b9d18**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:24:13 2018 +0530**

**m2**

**commit 1d527294c13721de544516ec9a248d4d96d5d419**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:23:47 2018 +0530**

**m1**

now go back to master branch by-

**git checkout master**

well if you listen carefully then you probably can guess what we can do right now to finally also add the changes to master because that;s the current status of master branch is m3. Latest commit both branch have in common is m3 commit. that is what we achived with rebase command. Now if in master we run –

git rebase feature

**git rebase feature**

**First, rewinding head to replay your work on top of it...**

**Fast-forwarded master to feature.**

now again git will have look at 2 branches and see what is alst commit both branches have in common, in our case this is m3 commit. then git will analyse our master and see what changed and that’s nothing. So there is nothing that git has to change internally. Then git will go to feature branch and see what changed there. It will find that we applied 2 addiitonal commits f1 and f2. Then it will apply these changes to master and after that it would add the changes that we apply to master and and as there are no changes, we will basically our master should now have m1,m2 ,m3 an dthen f1 and f2 commit. lets run git log-

**git log**

**commit 5b4bd3c50b4ee8aab788dca17b84071f42c6b4b4 (HEAD -> master, feature)**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:51:41 2018 +0530**

**f2**

**commit 52407674d3c99e87564aa00d52569816da15c213**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:25:33 2018 +0530**

**f1**

**commit 531ee3b0e309b30bb81936b8a96c0ede28b9e2c1**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:26:33 2018 +0530**

**m3**

**commit 84ddfe74400d64f61216f5f200b877f0436b9d18**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:24:13 2018 +0530**

**m2**

**commit 1d527294c13721de544516ec9a248d4d96d5d419**

**Author: unknown <sumitsood3127@gmail.com>**

**Date: Thu Oct 18 07:23:47 2018 +0530**

**m1**

on top we can see that we can see that we have common head of our master and of our feature branch. So these were 2 exmaples to show how we can use git merge and git rebase. As you can see both commands work differently, merge kind of combines information into one single commit at the end , if you use this squash command where **git rebase** will simpley as name says changes the base of our your code as we saw in feature branch, we suddenly did’nt start with m2 commit anymore but with m3 commit. this also brings us to kind of disadvantages of rebase command. You should never use rebase in public repositories, using them locally on your machine should be fine , so you work on your own projects. In bigger teams you can use it but it can cause serious damage in the end and lot of work for your colleagues. I wnt dive into all the details right now. But if you look at official docs, you cn see-**do not rebase commits that exists outside your repositiory.** Now I wnt dive deeper into this in this video as it is just meant to be intro video to show you capabilities of git merge and git rebase. But if you want to use it togather with your colleagues in public repositories, make sure to have a look at this article because otherwise this could cause some problems.