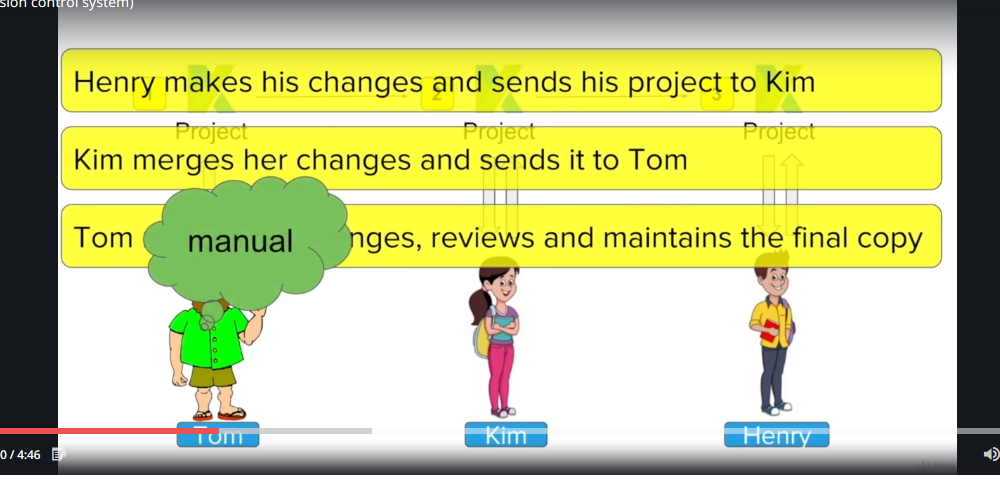
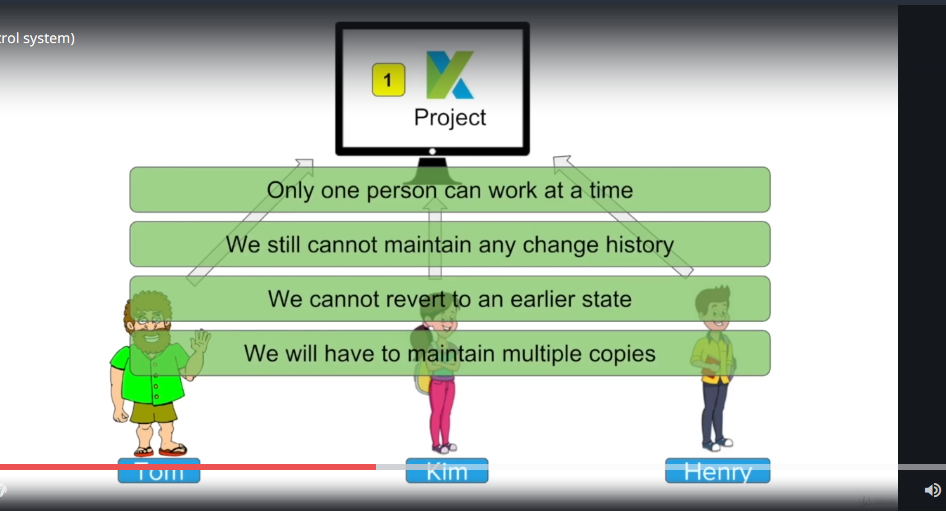
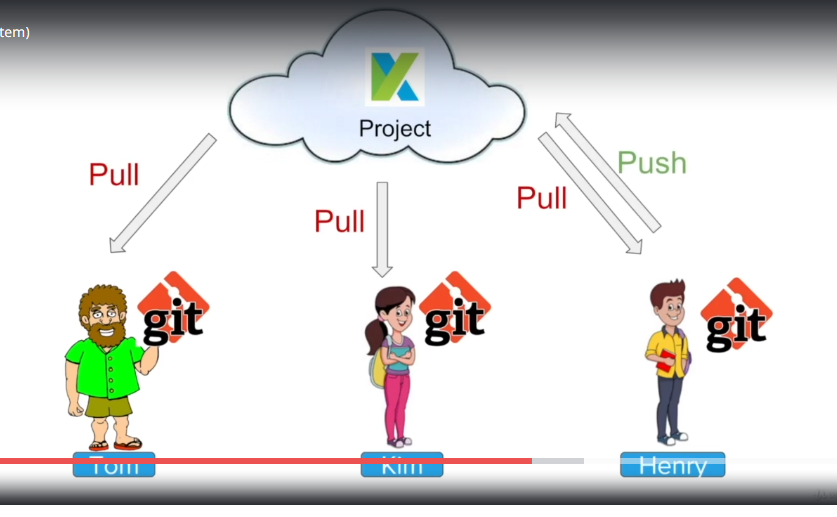
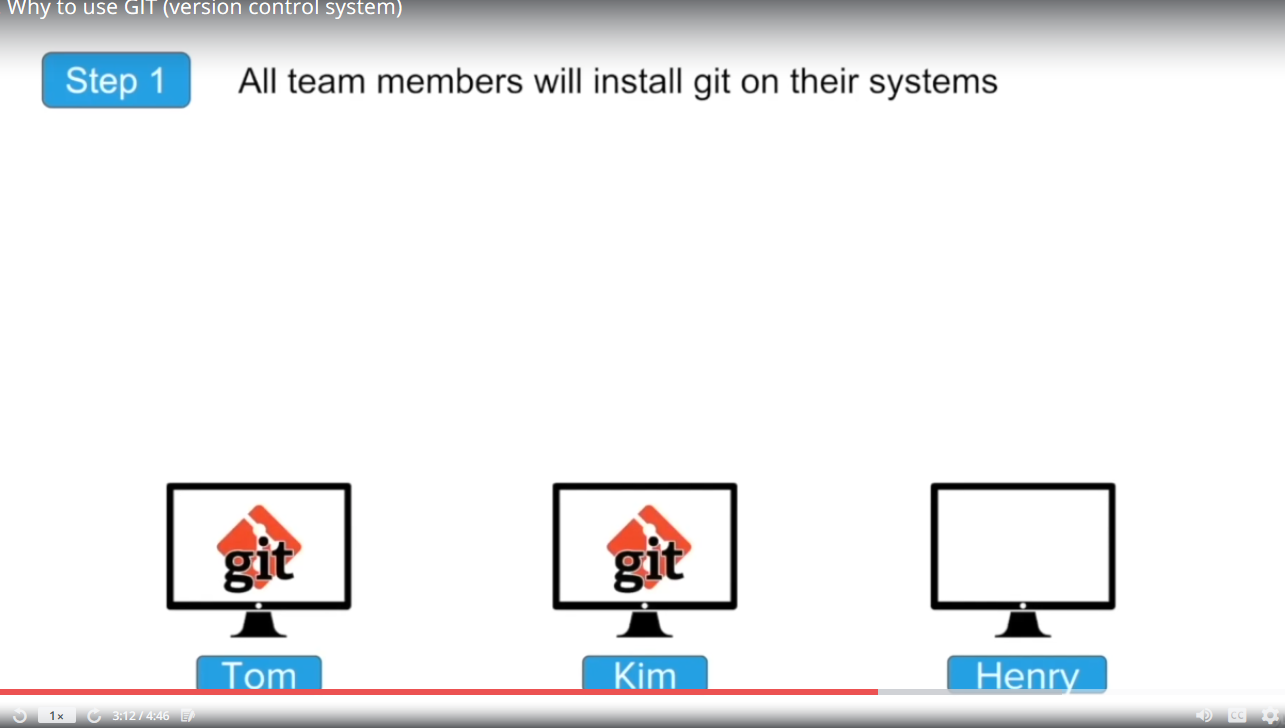
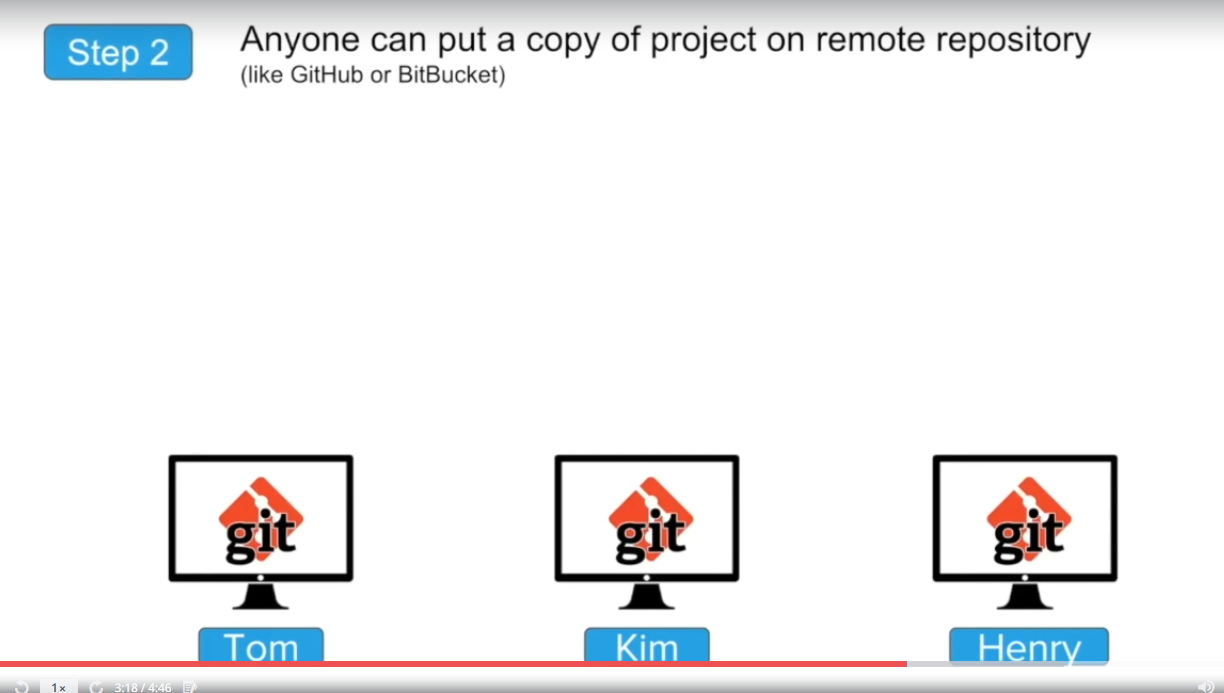
1.Version control system

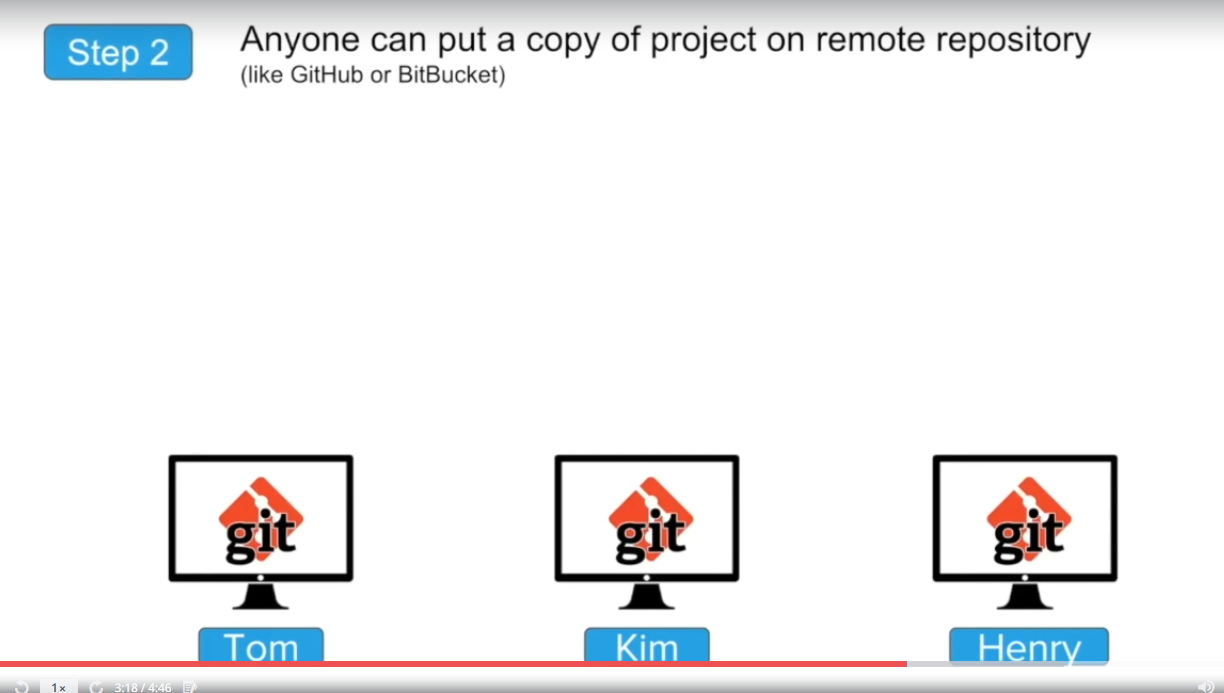












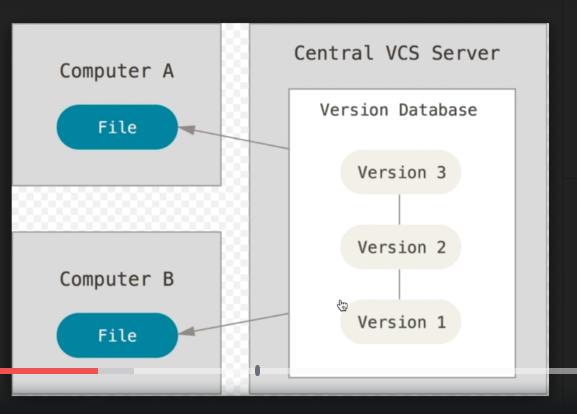
GIT:

Version control system basically maintain the history information about the evolution of the file or document (version1, version2, version 3)

In project suppose if we want to track all the changes done time to time we have to go for a version controlling system.

Track changes in file and folders.

Collaborate in team

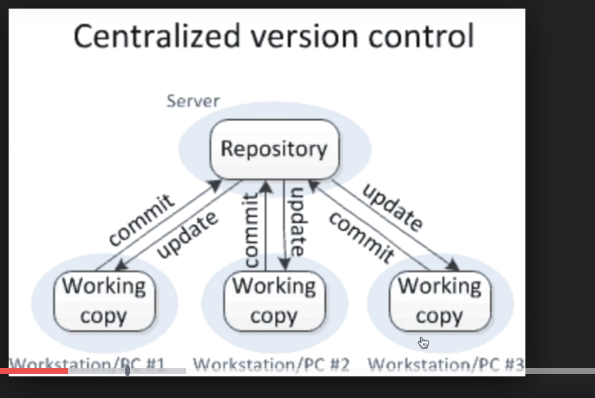


Version controlling system

1. Centralized version controlling system
2. Distributed version controlling system.

There will be a central repository on the server all the team members have to connect to this repository in order to collaborate with the team and all the checking checkout has to be done on this central repository.

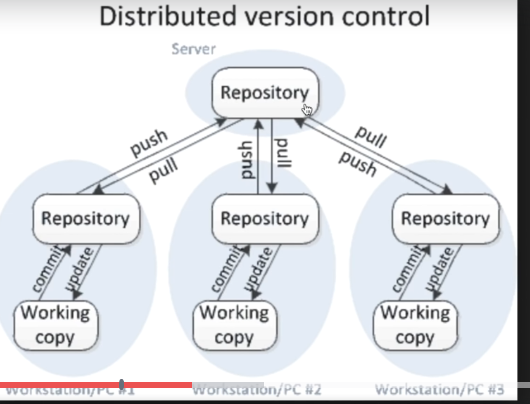
And system must be connected always to this repository



If anything goes wrong with the repository difficult to recover the content.

1. Every workstation can have its own copy of the repository
2. Advantage is if anything goes wrong to the central repository it can be recovered with the local copy from any of the machine.
3. Do not need to be online always. Work offline

Go online when push is needed.



GIT is distributed version controlling system

TFS is centralized version controlling system.

GIT hub:

Website to upload your repository online

Provides a backup for the repository

Provides Visual interface for the repository.

Makes collaboration easier with team members.

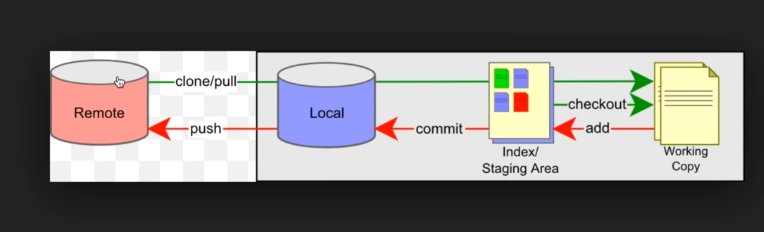
Are git and GIT HUB same answer is no GIT is version controlling system

GIT hub website to upload repositories for collaboration.

We can use any version controlling system with the GIT Hub

Similarly we can use different repository management system with GIT (like bit buckets)

Work flow:



Make pull or clone command to get the copy of the repository from the version management system like git hub

All the files will be present in the Index or staging area

Checkout a version from the staging area it will make a local working copy.

Add this changes back to the staging area

Commit this changes to the local repository

Finally push the request back to the repository management system GIT hub or bit bucket.

Installation:

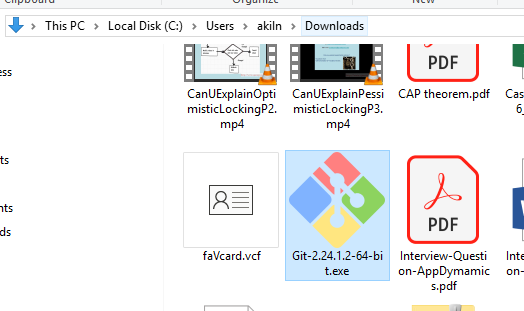
Step 1. Check if the git is already installed in your system.

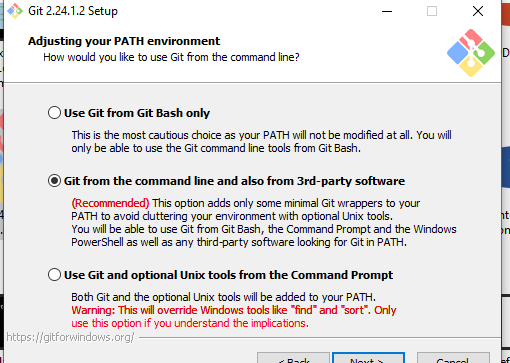
Git – version

Gives the information about the installed version.

Step 2:

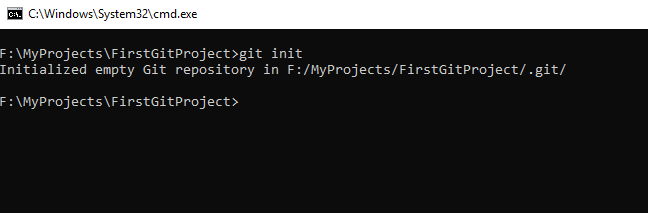
Download and install git



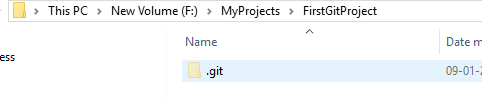


Create Project in the system and initialize the repository using command

**>git init**

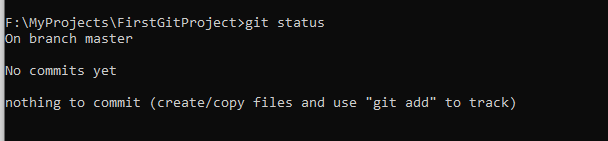


Create a hidden folder



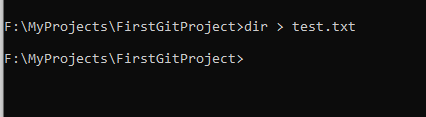
Get the repository state at any point of time run a command

**>git status**



To create a file

**Dir> test.txt**



Command to add new file to working directory

**$ touch file.txt**

Command to add files to the staging

**$git add . (adds everything)**

**$git add \*.\* (adds everything with extension)**

**$git add name of the file**

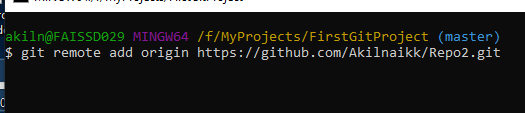
To commit the files from staging to the repository we need to run

**$git commit -m ‘Message’**

-m option indicates the message to be added at the time of commit (describes why the commit is done.)

**Adding to remote repository on the Git Hub:**

**$Git remote add origin /url of the repository/**



To commit

**$Git commit -u origin /branch name/**

Global properrty setting for the git

**$git config –global user.email ‘’test@test.com”**

**$git config –global user.name “TestName”**

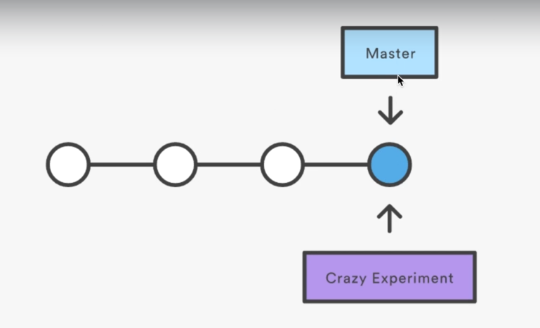
**$Git log** all activity log

**$git –help** all the commands used by the git

**Creating a branch:**

**$Git branch ‘Name of the branch’**

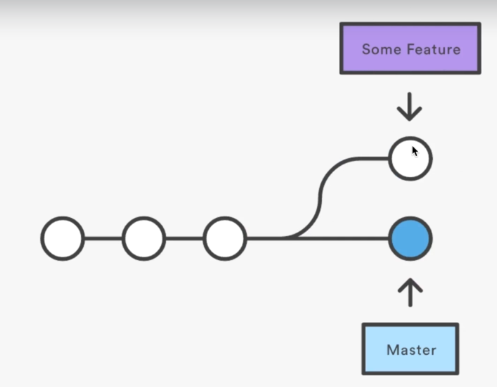
**$ git checkout -b repo2 create new branch names repo2 and checks out it**



Creates a new branch and put the pointer still point towards the current directory.

To start working on the branch we need to checkout the branch

**$Git checkout ‘Branch name’**



Merging branches:

**$git merge “Branch name”**

To do this first you have to checkout the branch to which you want to merge.

To remove or delete the branch

**$git branch -d ‘name of the branch’ -- delete from local**

**$git push origin –delete ‘branch name’**

**Sending mail from GIT:**

Go to repository

Then Select setting tabs from the tab

Old Integration and services in the newer version got to Notification a ns add email services.

GIT Tagging:

Process In which we create a special point in the history of the repository. Basically, used in marking release points. Like v1.0, v1.1…..

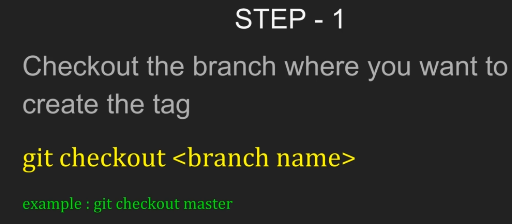
Why:

1. Point release point of the data or application
2. To create historic restoration point.
3. Historic point which can be referred in future.

Steps to create tags:

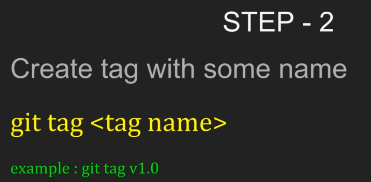
1. Checkout the branch where you want to create a tag

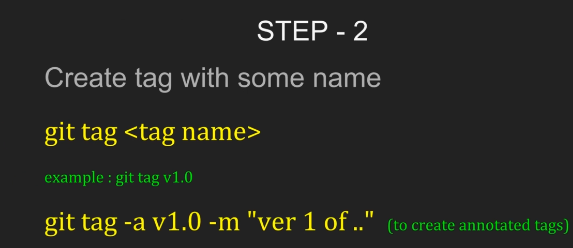
$Git checkout branchname



1. Create a tag for the repository

$git tag ‘tag name’

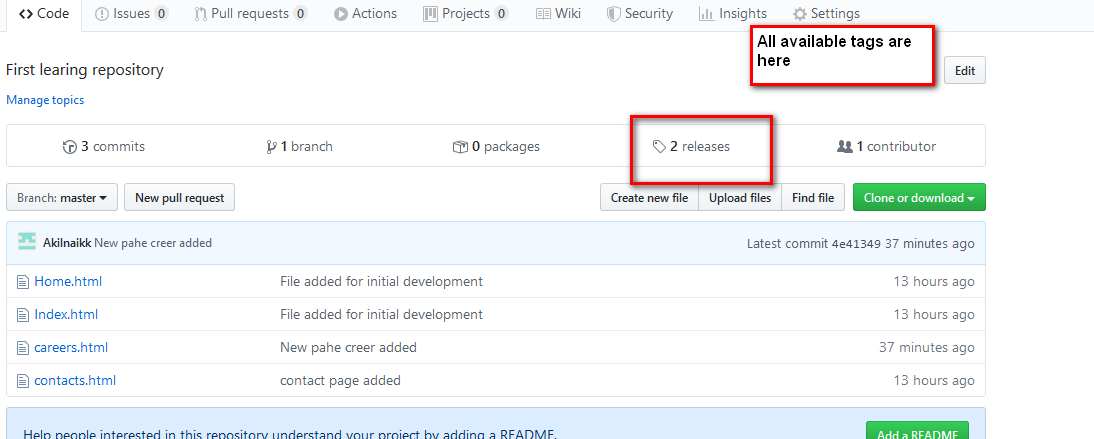




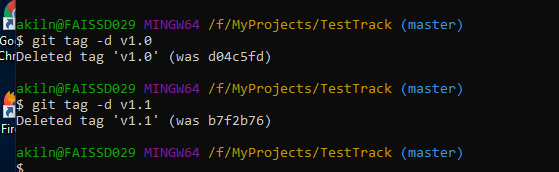
We also need to push the tags to the remote repository using command

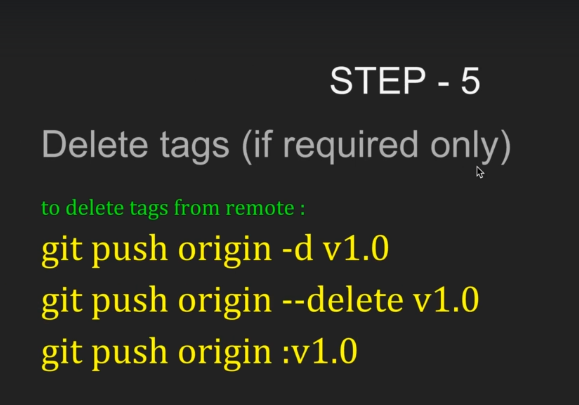
$git push origin ‘tag name’

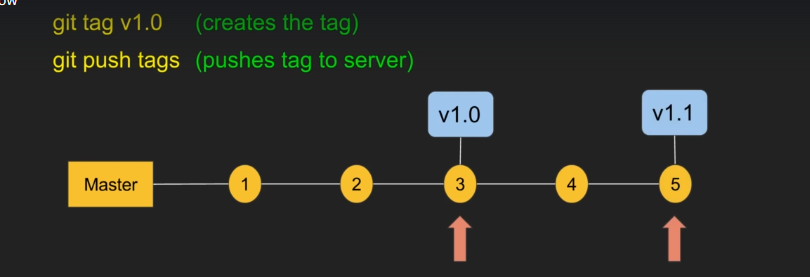
$git push –tag all tags will be pushed to the server



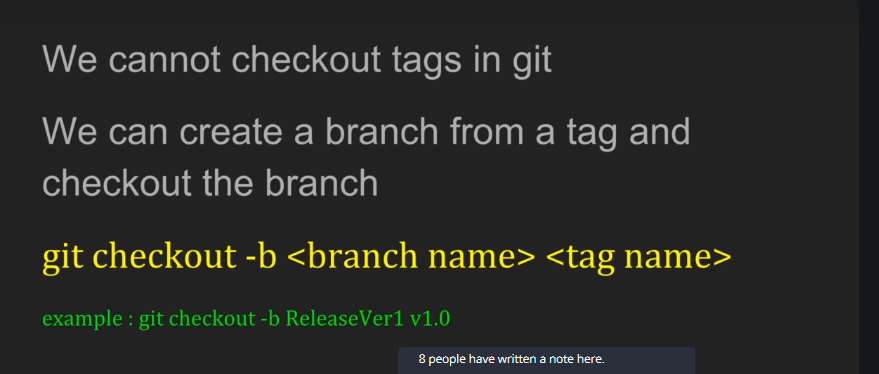
TO remove the tags







Checkout the branch from the tag point



This is used when you want to make a branch from particular tag

**Git branching vs Rebase:**

Bothe methods used for integrate the changes from one branch to the another branch.

**$ git Merge ‘from branch’**

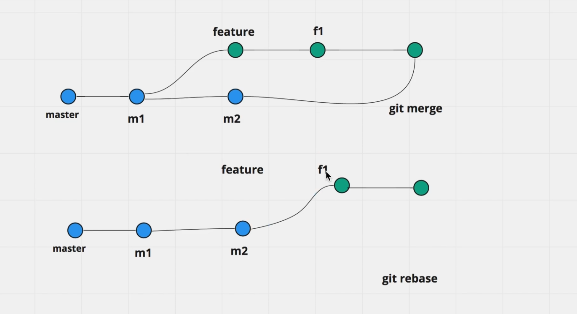
Before checkout the branch where you want to merge



Git rebase:

$git rebase Master

In this process we are placed in current working directory and do a rebase of master that will move the branch to the tip[ of the master branch.



Terminologies:

Master

1. Main/default branch created when making a new repository.
2. Single point of truth used with CI tools for deployment.
3. Every repository has master branch that by default follows the remote repository master.

Branch:

1. Light weight working copy for the parallel line of development.
2. Staging area, working directory, project history.
3. Work without impacting the parent and finally merge the changes back to the parent.

Head:

1. Pointer that point towards the last commit from the working branch
2. Per repository there will be one head.
3. After commit head will move towards new commit point
4. On switching branch head will move to the last commit from the branch.

Remote:

1. Repository on the network or the cloud outside the local machine.
2. More than one remote pointing from the local.
3. Can be managed and reference with short names.

Origin:

1. Associate remote repository with names.
2. Logical name set for the default repository.
3. Refer a repository with logical name instead of the entire url

Clone:

Copy the repository from the remote to local machine.

Gets entire repository to your system.

Push:

Write the changes from the local repository to the remote repository. Before doing that changes need to be committed in local.

Pull:

Command used to get the changes from the remote repository to the local machine

Syn from the remote repository to the local and keep your copy up to date

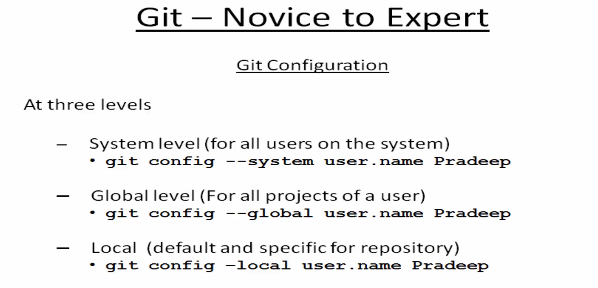
Merges the local with the server version.

Fetch:

Update local version upto date from the server version.

Get the server version of the data but will nor merges the changes with the local version

Explicitly ned to perform merge with the local version.



System: For all the users working on the system

Global level: for the user working on the different repository

Local: At the repository level. (user working on multiple projects )

To see the setting

**$git config – global –list**

**$git config – global user.name**

**$git config – global user.email**

Methods to ignore the files for staging

**$touch .gitignore**

New file will be created now add all the files inside this which you want to ignore basically.

Suppose if you use a regular expression to map the ignore file list and if you wat ignore any from the matching list use

! file name

Insdie the .gitIgnore file

After the add say you want to remove items from the commit for this we need to run

**$git checkout – file name (ignore the changes done to the file)**

Add version for staging:

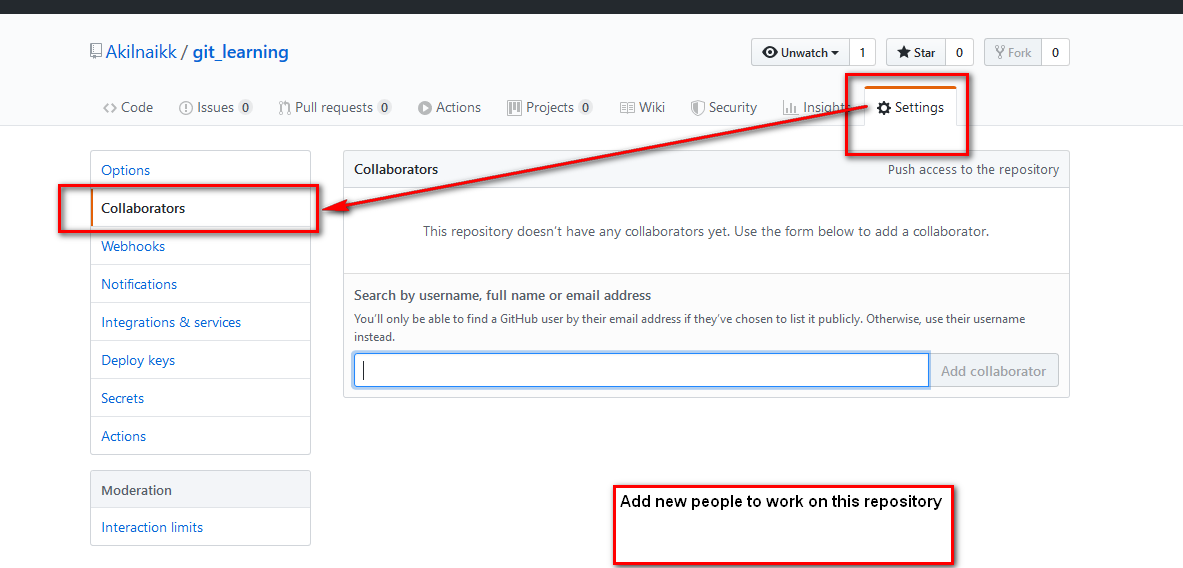
**$git add .** (to add new and modified but not delete)

**$git add \*.\***

**$git add – u** (add modified and deleted but not new)

**$git add – A** (modified deleted and new)

Adding collaborators to repository:



To see the difference in changes to the file

**$Git diff file name**

To delete the file inside the repository

**$Git rm file name**

TO get the deleted file rom staging area to the working directory:

**$git reset HEAD filename**

**$git Checkout – file name**

To get the log info

**$git log**

**$git log -2 (last two action)**

**$git log – pretty=onelie (display log record in single line)**

**Clone/Copy a repository from the remote:**

**$git clone url of the file in the location.**

To find the origin for the repository

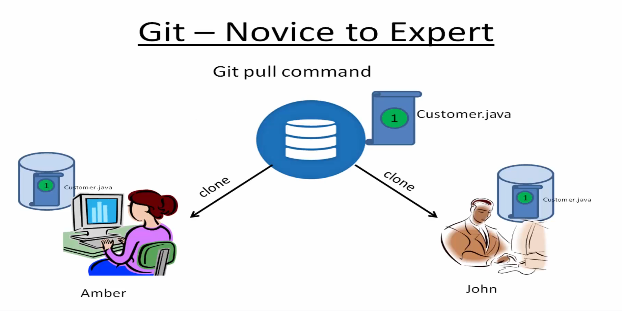
$git remote -v get the lit of all remote repository’s.

Executed once initially when you want to setup the repository in local by copying the repository from the remotes.

**PULL:**

Say periodically you want to update the local repository with the content from the remote repository.

We nee to go for pull command to keep your repository up to date.



**$git pull origin master**

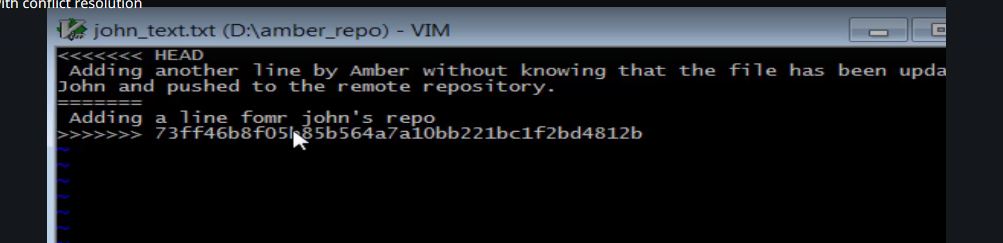
When changes done in the local repository is not committed and we try to pull a latest version from the remote repository the merge operation will be aborted.

Commit and add simultaneously

**$git commit -am ‘Comments’**

On pull if the local repos has content modified and pull was success it will not merge the changes success full it merges everything to you. Then manual developer has to merge his changes with remote version and then do a final push.

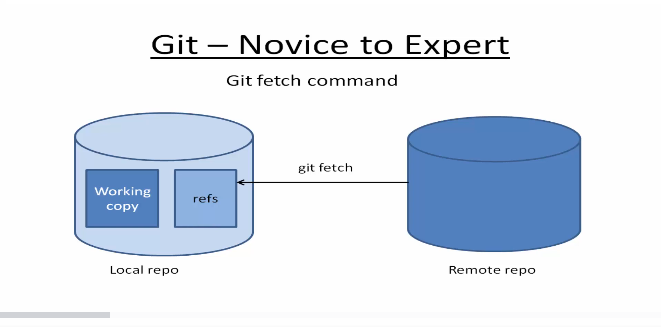
Merged file shows everything from local and remote repo as shown below

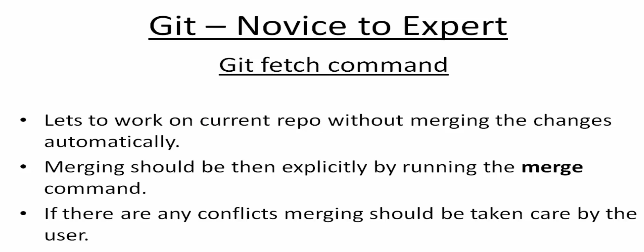


**Git fetch:**

Pull command is a combination of git fetch + git merge

Fetch command used to get remote version to local repository and it will not update his code base/working directory





**$git fetch origin master**

Get that latest version from the remote repository and puts it in remote repository tracking branch reference file will not merge it to the copy.

Finish the local development is done commit the local repository

then merge it with remote local tracking

resolve the conflicts

merge the changes to local repo

finally push the changes to the remote repository.

**$git merge origin/master**

**Branch:**

Mater is final default branch which is deployed to production

Never work directly on master branch

Create sub branch finish task test it well and finally merge to master

**$ git branch –list** list all branched in the repository

**$ git branch --list -r** list all branches in remote repository.

**$git branch ‘name of the branch’** create a branch with the specified name

Each branch has its own staging and commit area.

Need to checkout a branch in order to work on it

**$git checkout ‘branch name’** to switch between the branches

**$git checkout -b ‘branch name’** to make a new branch and switch to the new branch.

Merging changes

$git merger “name of the branch” to be merged to current branch.

**$git branch -d ‘branch name’**

**$git push –delete origin ‘branch name’**

Forceful detection of the branch use -D as option in the command.

**$git branch -D ‘branch name’**

Tagging in GIT:

Basically, used for the deployment purpose. Used to create a check point in your application.

Say a feature is developed and it is approved by the quality team for the release we create a tags on the current version of the code base.

Once the tag is created no more changes can be done or added by the test team on the same tag.

**$Git tag** ->list all the tags in repository

**$Git tag <tag name>** create tag in local repository

**$Git push origin <tag name>** push the tag to the remote server.

Delete tag

**$ git tag -d** <tag name>

**$git push origin :refs/tags/V1.0**

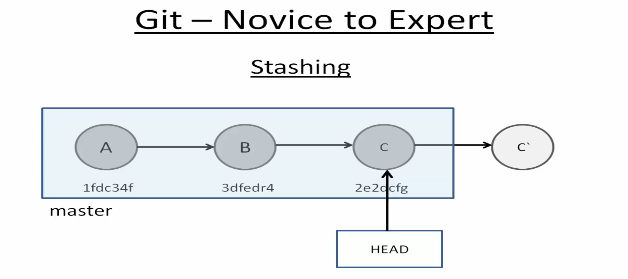
Stashing in GIT:

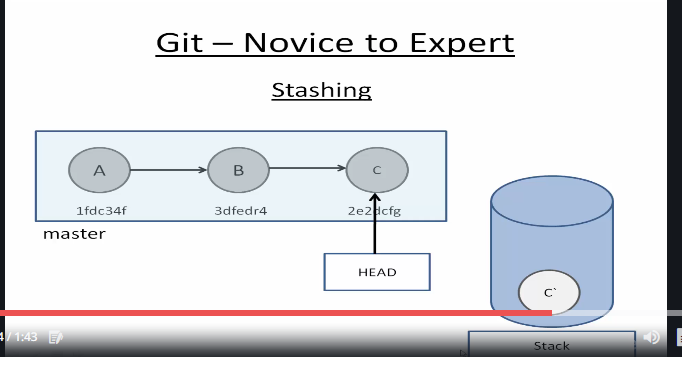
Move uncommitted changes to a temporary location.

Scenarios:

1. Working on some feature suddenly a production bug comes in. So need to stop the current work and proceed with bug once it fixed get back to your task.
2. Working on some wrong version and lost of changes have been done on that branch you need to save those changes to some location.

Can also be used to move uncommitted changes from one branch to another branch.



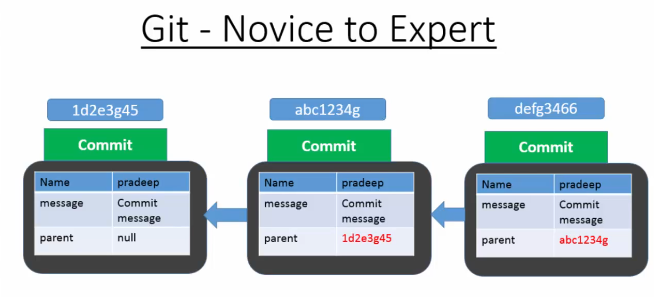


For stashing git uses separate queue to stash the changes which can be then removed in last in first out basis

**$git stash save “message for stashing”**

**$git stash list** 🡪 this will get all the changes stashed so far.

Commit chain:



Merging:

Consolidating changes from sorce branch to destination branch

1. Fast forward merge
2. 3 ways merge
3. Rebase

Fast forward merge:

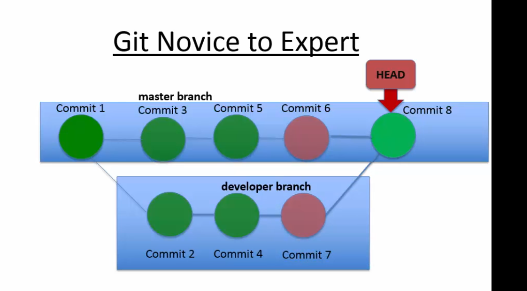
Make changes in both repositories. Be on the /checkout the repository to which you want to merge.

Then execute the merge command

**$git checkout master**

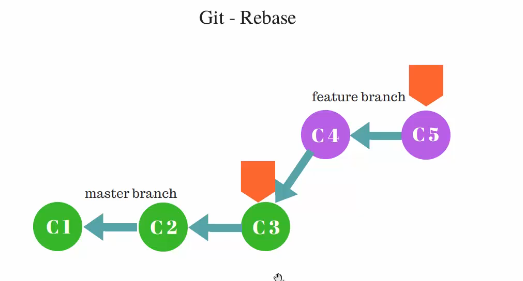
**$git merge <branch name>**

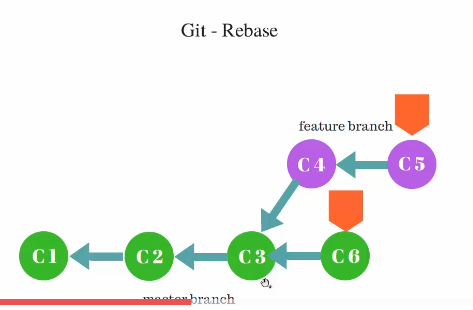
3 Way merging:

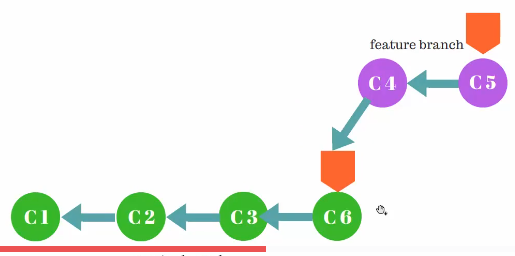


Merging is used in the scenario where in checkin is happening in the parent branch even if the child branch is created.

In such scenarios merging will result into new commit object and a head point is moved to the ne commit object created.







In merge method new commit object will be crated pointing towards last commit object from both the branches master and sub branch

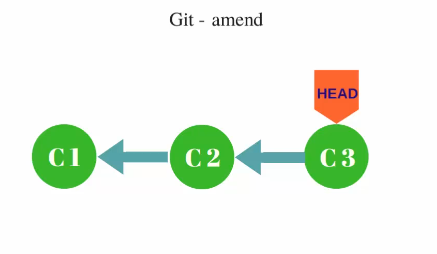
In got rebase the source branch will be stacked on top of the last commit from the destination branch

Be on the target branch where you want to merge all changes.

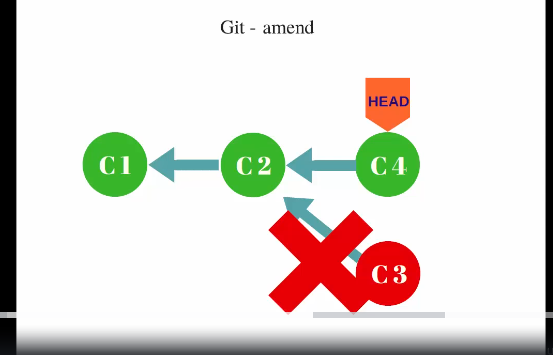
$Git rebase <branch name>

Git amend:

With this command we can modify the latest or last commit.



Ideally we cannot modify what is there in the commit but internally we can replace this commit object with the new commit object.



When we add a new commit with amend command old commit will be garbage collected by the git

Eg: while developing the changes there are few typo errors.

While developing feature important config file is missed out from the check-in

**$git commit –amend -m <comment for the commit>** (replaces the old commit)

**$git commit –amend –no-edit** (replaces old commit without changing the comment )

1. Only local commit can be amended.
2. Commit can not be amended on remote repository.
3. Amending will create a new commit object.
4. Old amend object will be garbage collected.

Git revert:

Basically used to revert the changes done in a particular commit

**$Git show commitid** will get the changes done on a particular commit.

**$git revert <commit id>**

**$git revert –abort**

**$git revert --continue**