**GitHub**

Tutorial 1:

Today we will learn

1. How to download and install git

2. Signup and create account on GitHub

3. Add a project/folder/file to git

4. Track and commit changes

5. Add the repository to GitHub

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Step 1 - Check if git is already installed: [0:32](https://www.youtube.com/watch?v=0Icla6TVNNo&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG&index=2&t=32s)

Step 2 - Download and install git: [1:52](https://www.youtube.com/watch?v=0Icla6TVNNo&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG&index=2&t=112s)

Step 3 - Signup and create an account on GitHub: [3:40](https://www.youtube.com/watch?v=0Icla6TVNNo&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG&index=2&t=220s)

Step 4 - Add your GitHub email and username to git: [4:50](https://www.youtube.com/watch?v=0Icla6TVNNo&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG&index=2&t=290s)

Step 5 - Add file/folders to git - tracking: [6:44](https://www.youtube.com/watch?v=0Icla6TVNNo&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG&index=2&t=404s)

Step 6 - Commands: [8:43](https://www.youtube.com/watch?v=0Icla6TVNNo&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG&index=2&t=523s)

**Converting the container to new image:**

docker container commit <container id> - it will create new image of our existing container and it is not having any name, it is having only new image id.

docker image tag <new image id> <name of the new image with tag> - for giving name to the new image, which is having the without name.

**Uploading Image to the docker hub:**

docker login – it is for login to the docker hub command, and it will ask username and password.

docker push <image name> - it is command for uploading our new image to the docker hub.

#############################################################################

#############################################################################

Head – Head is a pointer to the branch (head is the pointer to the last commit made)

Tree – git area currently where you’re working on (Un-staged changes)

Index – is the place where we place the files that you want to commit. (it is a staging area between Working tree and repository.)

Git fetch: it will retrieve the latest metadata info from origin

Git merge: it will merge the latest changes into existing code

Git pull: it will download the latest changes from origin (git pull = git fetch + git merge)

Git reset – it will remove the files from local repo (staged area)

Git rebase - it merges the entire history of two branches into one. (Rebasing allows you to rewrite the history of a Git repository.)

Git rebase -i - The -i flag is used to begin an interactive rebasing session. This provides all the benefits of a normal rebase, but gives you the opportunity to add, edit, or delete commits along the way.

Git add - Moves changes from the working directory to the staging area.

git branch - This command is your general-purpose branch administration tool. It lets you create isolated development environments within a single repository.

Git checkout - In addition to checking out old commits and old file revisions, git checkout is also the means to navigate existing branches.

Git clean - Removes untracked files from the working directory. This is the logical counterpart to git reset, which (typically) only operates on tracked files.

Git clone - Creates a copy of an existing Git repository. Cloning is the most common way for developers to obtain a working copy of a central repository.

Git commit - Takes the staged snapshot and commits it to the project history. Combined with git add, this defines the basic workflow for all Git users.

Git commit --amed - Passing the --amend flag to git commit lets you amend the most recent commit. This is very useful when you forget to stage a file or omit important information from the commit message.

Git config - A convenient way to set configuration options for your Git installation. You’ll typically only need to use this immediately after installing Git on a new development machine.

Git init - Initializes a new Git repository. If you want to place a project under revision control, this is the first command you need to learn.

Git log - Lets you explore the previous revisions of a project. It provides several formatting options for displaying committed snapshots.

Git merge - A powerful way to integrate changes from divergent branches. After forking the project history with git branch, git merge lets you put it back together again.

Git push - Pushing is the opposite of fetching (with a few caveats). It lets you move a local branch to another repository, which serves as a convenient way to publish contributions. This is like svn commit, but it sends a series of commits instead of a single changeset.

Git reflog - Git keeps track of updates to the tip of branches using a mechanism called reflog. This allows you to go back to changesets even though they are not referenced by any branch or tag.

Git revert - Undoes a committed snapshot. When you discover a faulty commit, reverting is a safe and easy way to completely remove it from the code base.

Git status - Displays the state of the working directory and the staged snapshot. You’ll want to run this in conjunction with git add and git commit to see exactly what’s being included in the next snapshot.

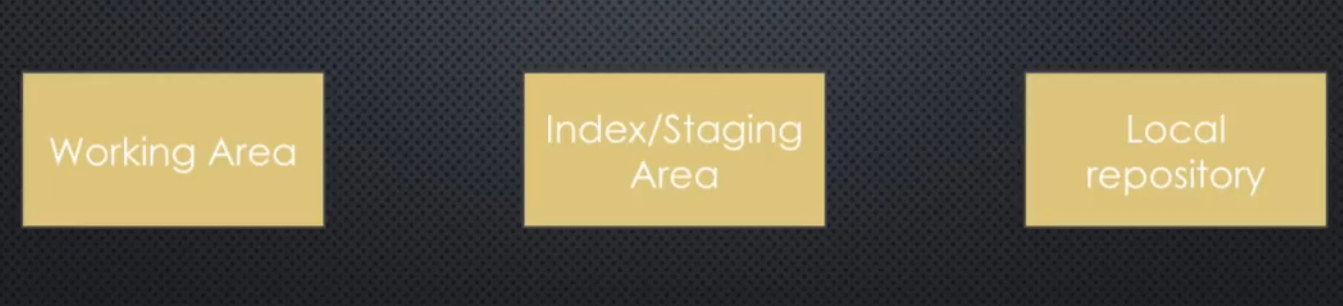
git checkout -b iss53

git branch iss53

git checkout iss53

##################################################################################

##################################################################################



git --version

git config --global user.email "yourGitHub@email.com"

git config --global user.name "yourGitHubusername"

git reset --hard – it will reset the code committed in the local git repo.

git init – it will create local git repo.

git status – it will check the status of current git local repo.

git log – it will show commit history of current repo.

git add . – it will add entire folder data into staging area.

git add <filename> - it will add file into a staging area.

git commit -m “message” – it will commit entire staging area code to local git repo.

git commit -a -m “message” – it will commit entire staging area code to local git repo.

git remote add origin <git URL> - it will add the git URL as a origin.

git push origin master – it will push entire local git repo code into remote git repo.

git push -u origin master – it will push entire local git repo code into remote git repo with upstream reference.

git push -f origin master – it will push entire local git repo code into remote git repo with forcefully.

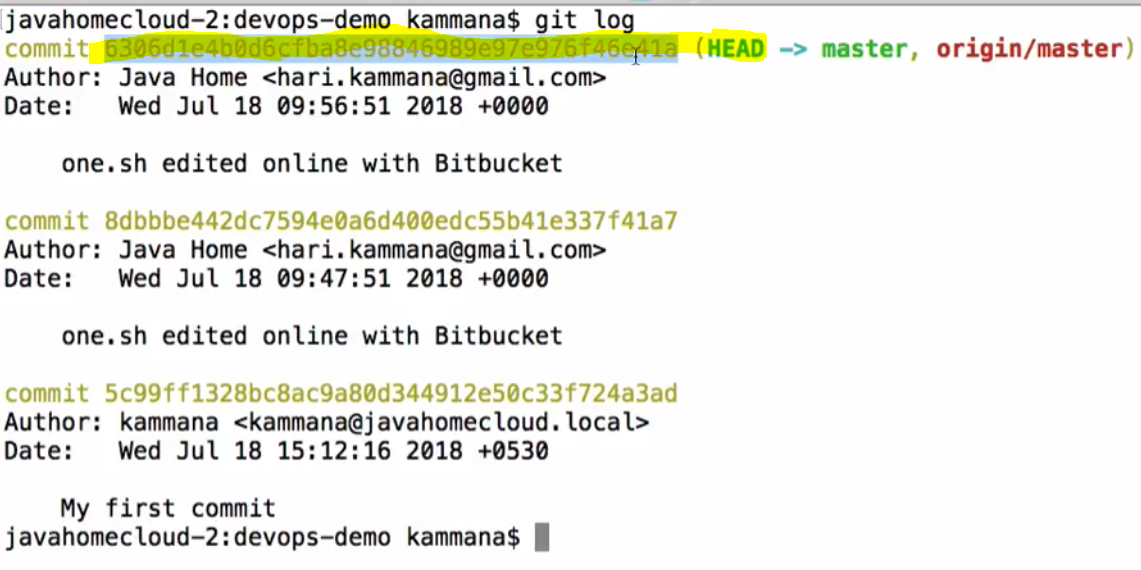
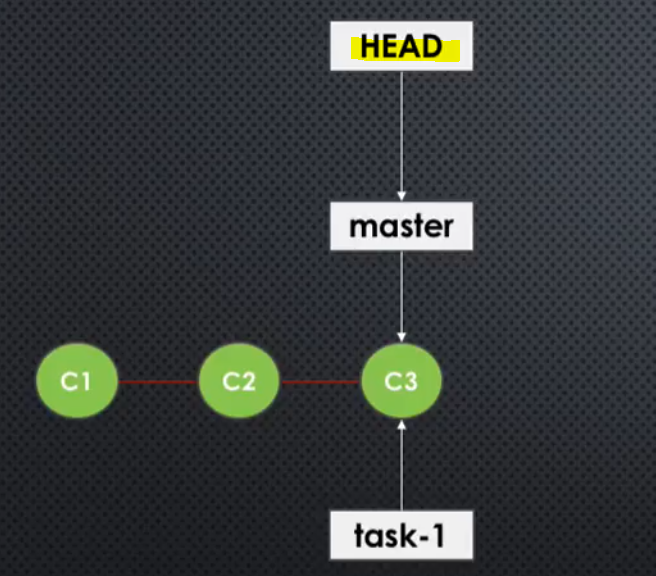
git remote -v – it will display the origin URL.

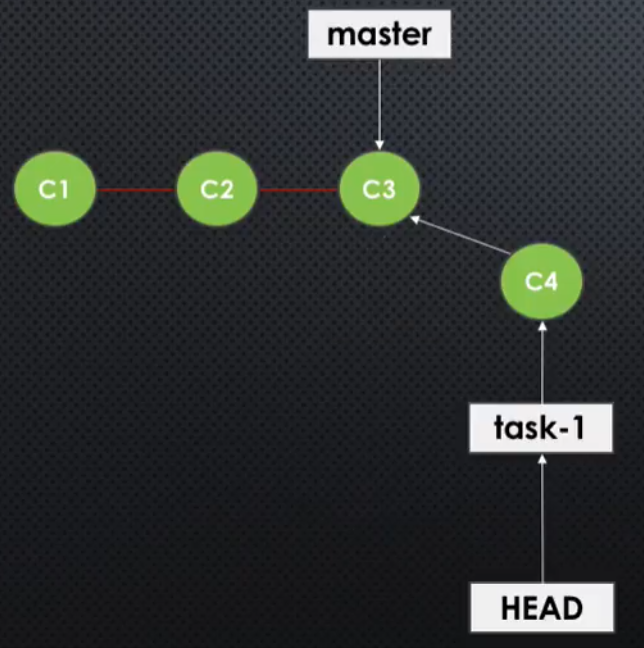
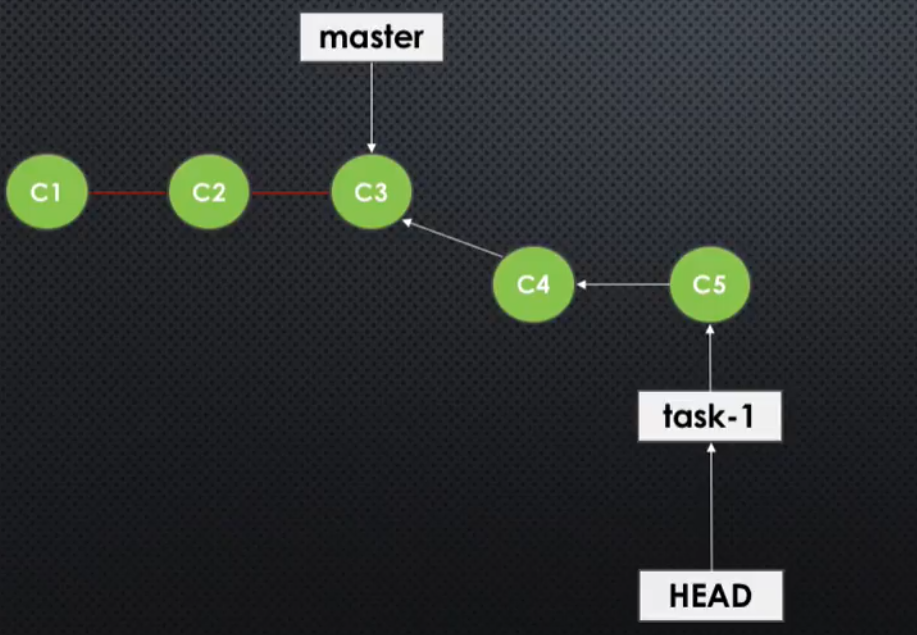
git fetch origin master – it will fetch the new changes into git local repo but it will not merge the code. (fetch new changes into local git repo metadata from remote origin)

git merge – it will merge the code into local git repo. (merge new changes into local git repo from local metadata)

##################################################################################

git head – it is a previous last commit.

git branch – it will give isolation to your work.

git branch task1 – it will create new task1 branch.

git branch – it will show you branches which are available.

git checkout task1 – it will switch your branch.

git merge task1 – it will merge all changes in task1 branch to master. (while executing this one we should present in the master branch.)

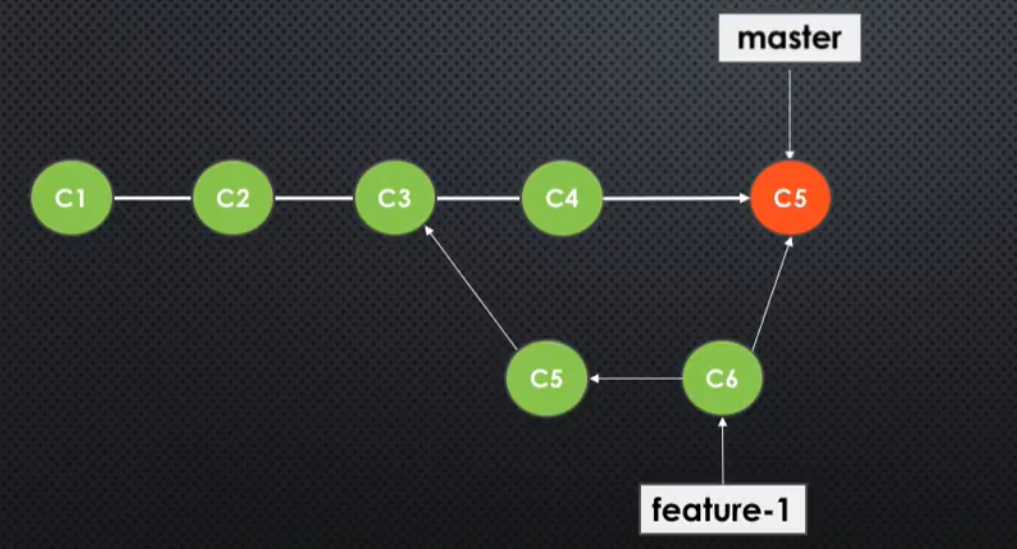
git branch -d task1 – it will delete the task1 branch.

(while deleting the branch need to ensure it should merge with main branch of remote repo or else you will get error.)

git push origin task1 – it will push local branch to the remote repo.

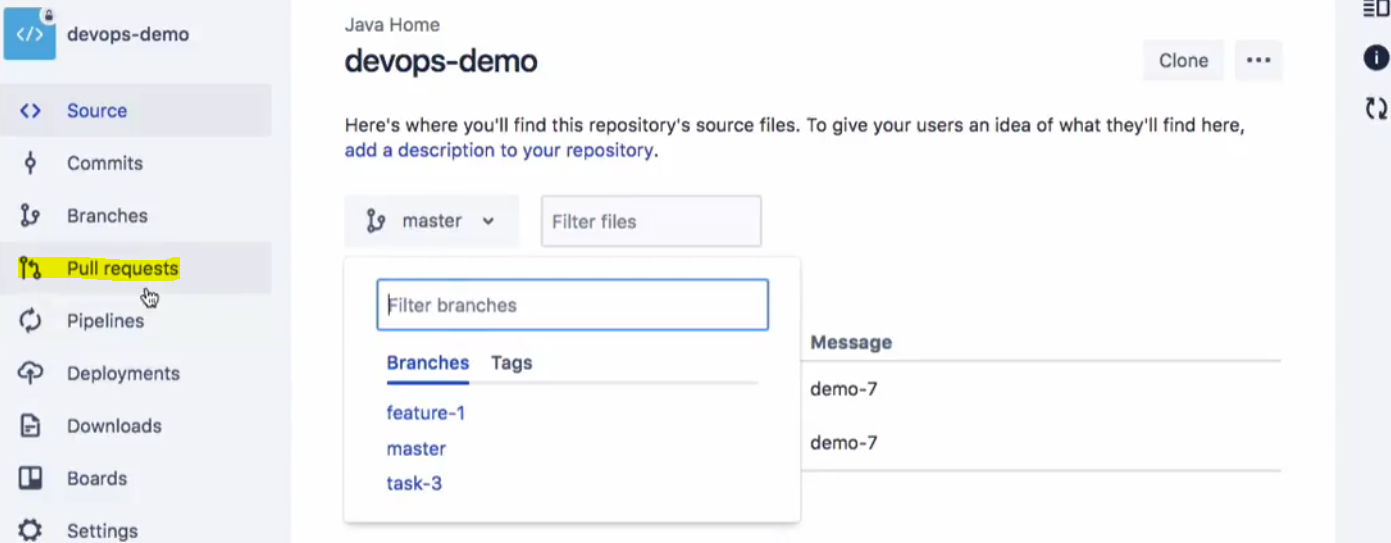
##################################################################################

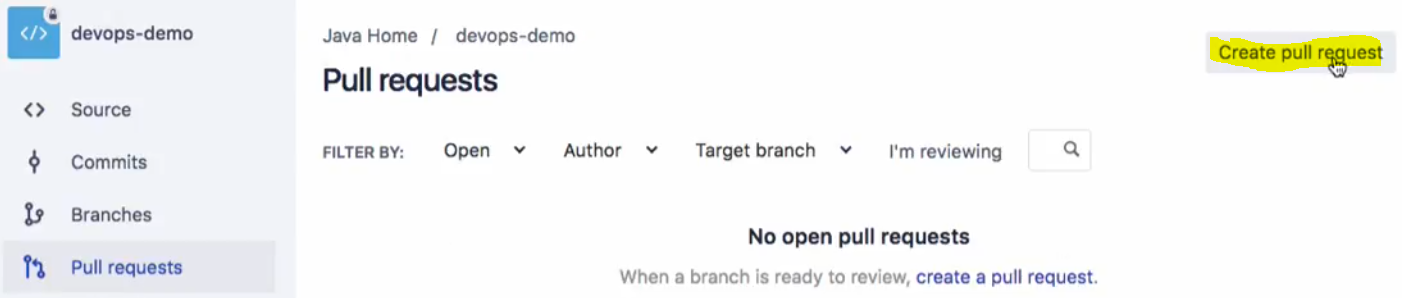
git pull request – it will create request to review the code before committing into the master.

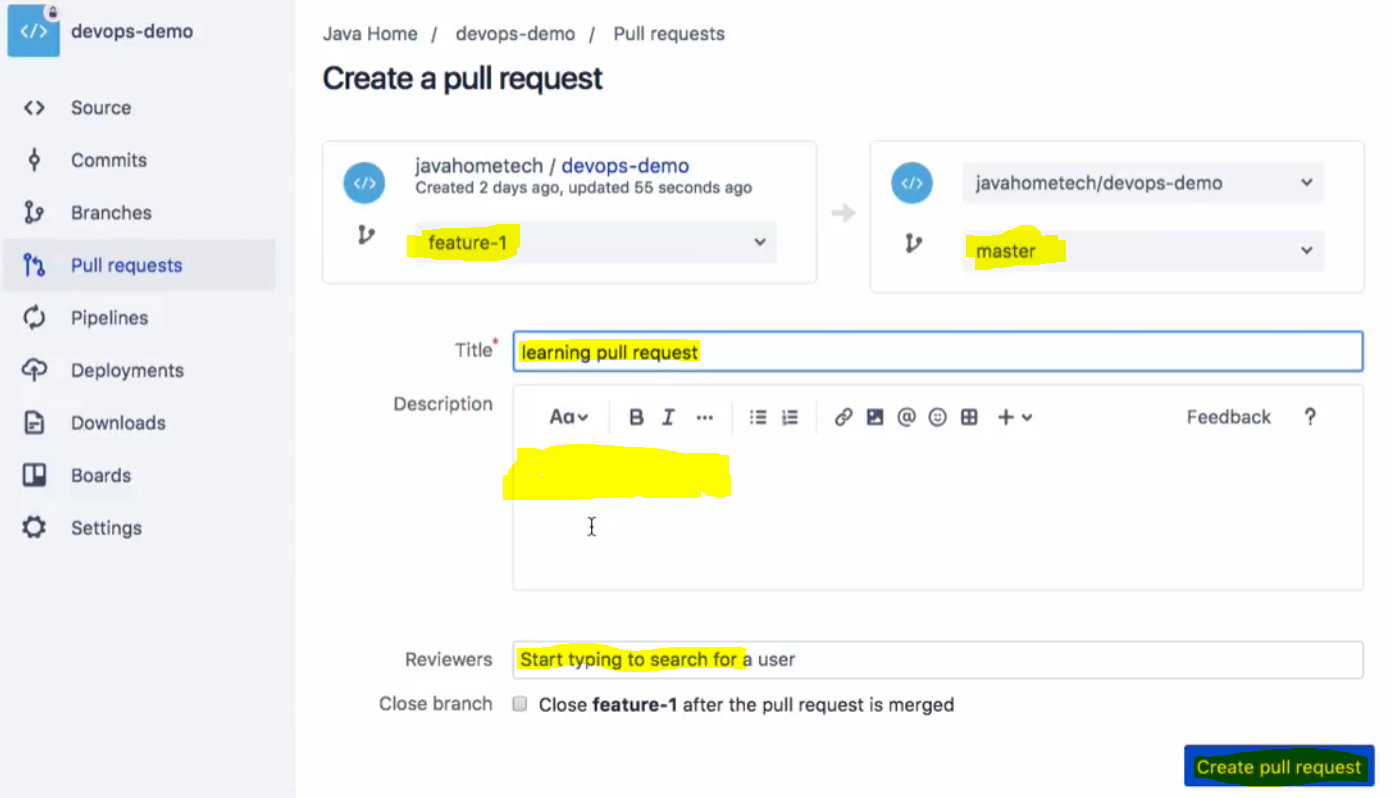


1st we need to push local repo branches into remote git repo.

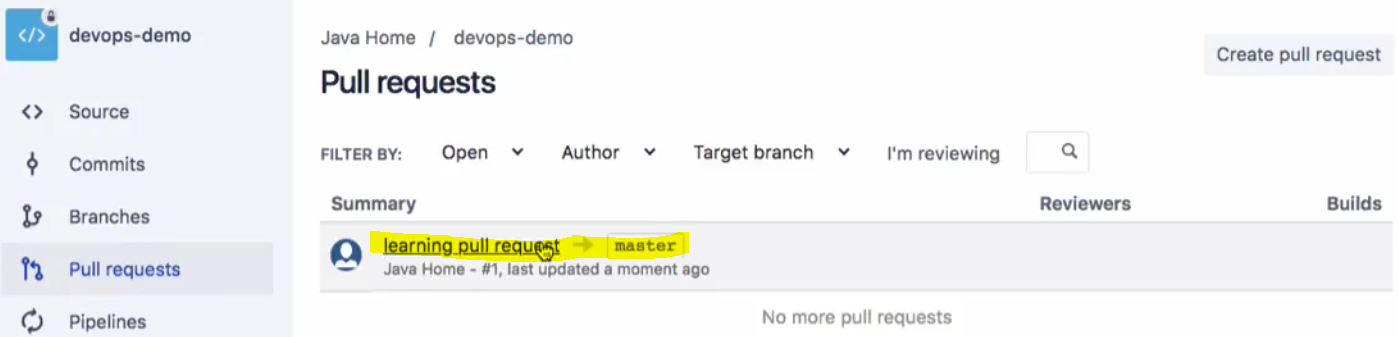
Then follow these steps:







Need to put Source branch and destination branch, title, description, and reviewer’s name.



They will get pull request.



They will get this modified option to review the code and they need to approve the code, then you will get option to merge into master.

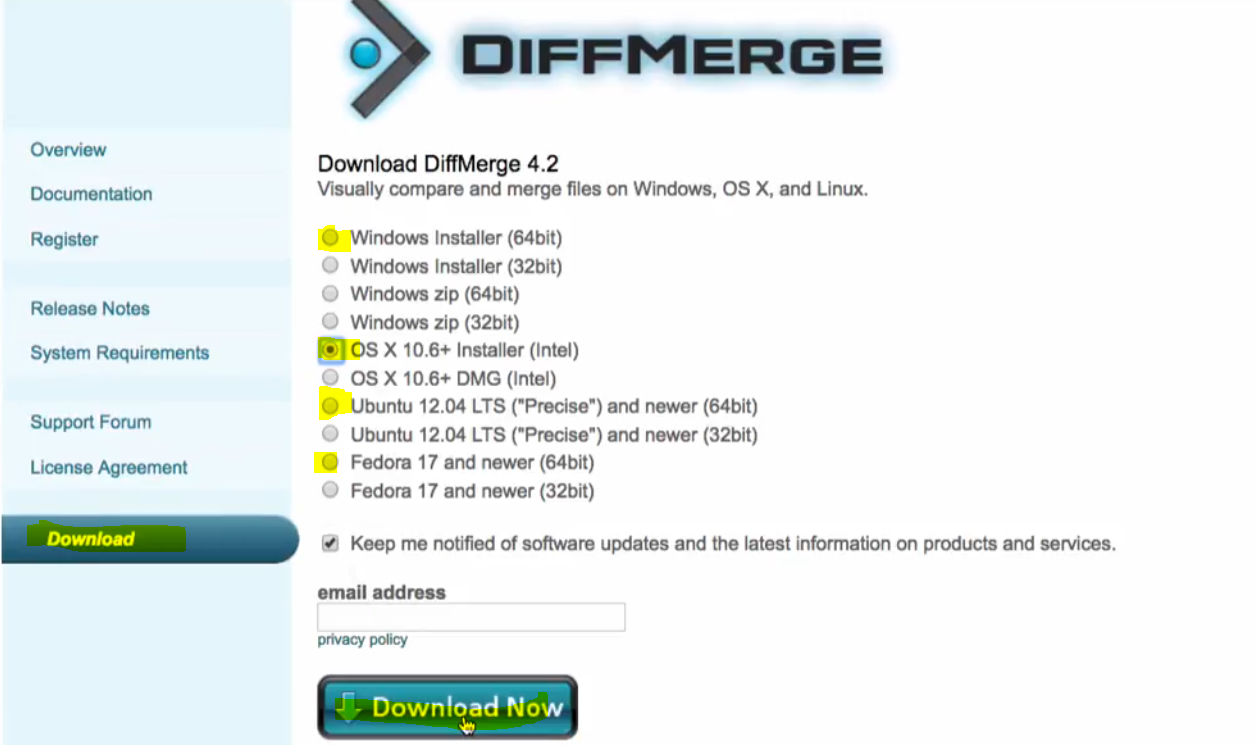


Here we need to merge.

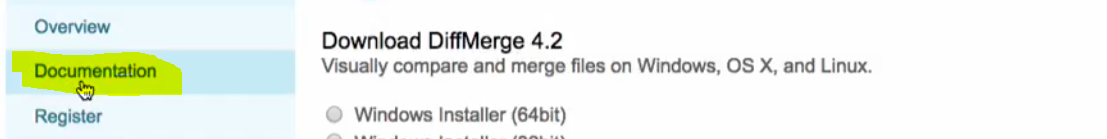
##################################################################################

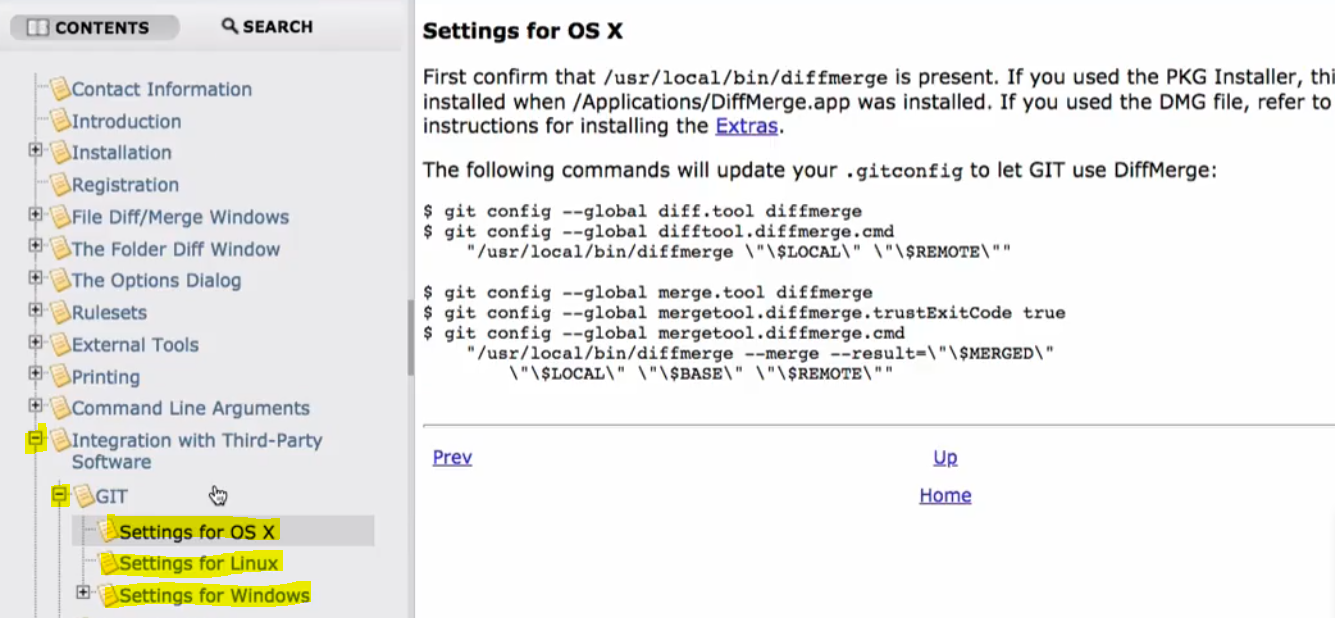
diffmerge – it will resolve conflict of the code. It is third party tool, need to configure with git.

For Configuration:



After the download





##################################################################################

git checkout . – it will discard all the changes in the working area.

git checkout <filename> - it will discard only the specific file changes in the working area.

git reset HEAD \* - it will un-stage all the staged files.

git reset HEAD <filename> - it will un-stage specific file from the staged file.

git reset HEAD~1 – it will remove the latest commit code from git local repo, and it will keep the code into the working area.

git reset HEAD~3 – it will remove the latest 3 commit code from git local repo, and it will keep the code into the working area.

git reset --soft HEAD~1 – it will remove the latest commit code from git local repo, and it will keep the code into the staged area.

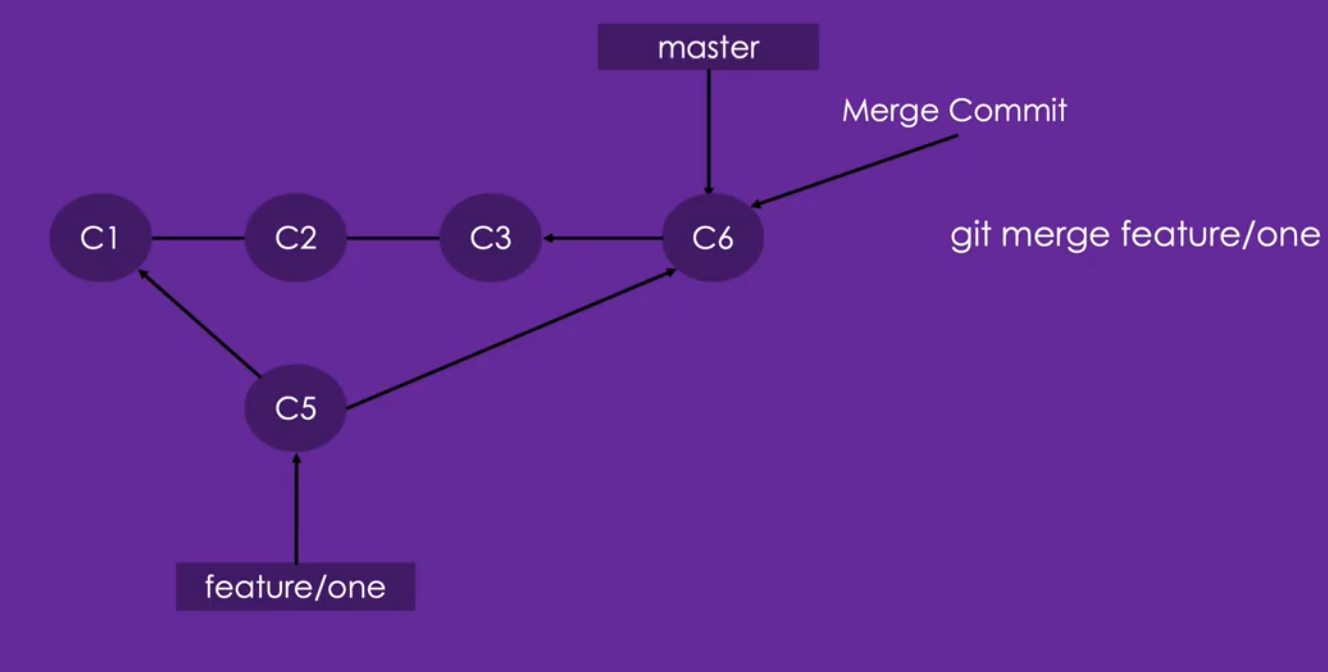
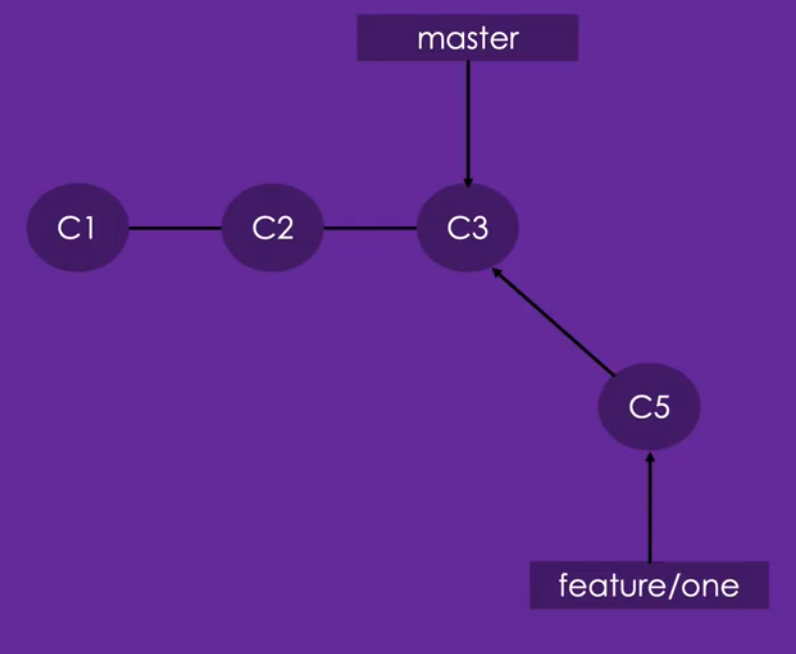
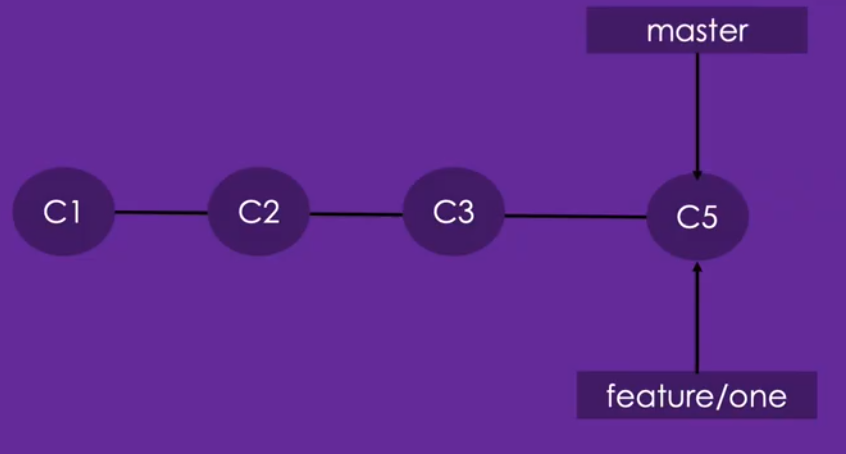
git reset --hard HEAD~1 – it will remove the latest commit code from git local repo, and it will not keep in the anywhere.

git revert <commit id> - it is useful for remove any of the commit instead of latest commit.

it will remove all changes commits using commit id and it will make new commit for that. It will not change any commit history.

##################################################################################

git rebase – it will keep history linear.

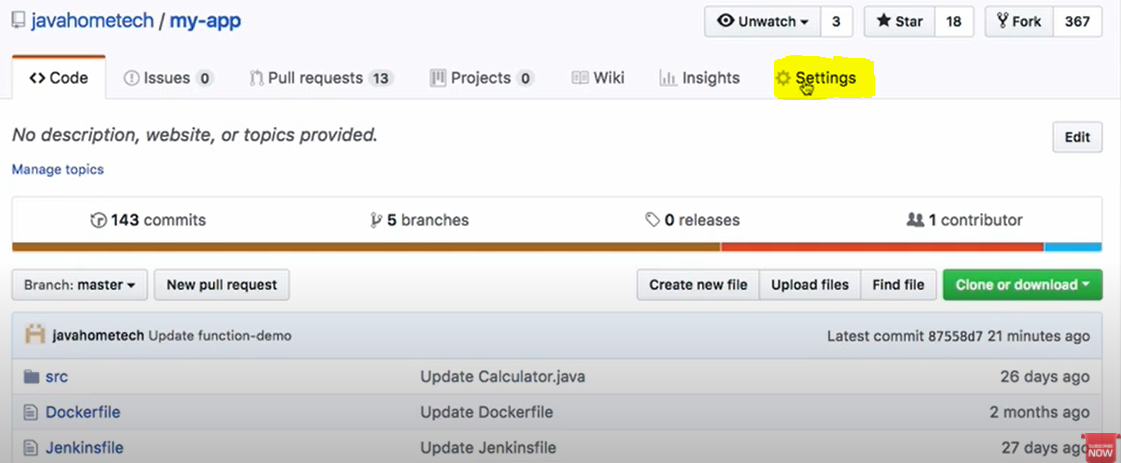
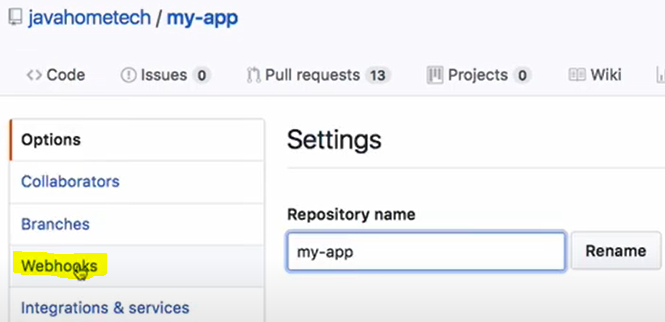
  

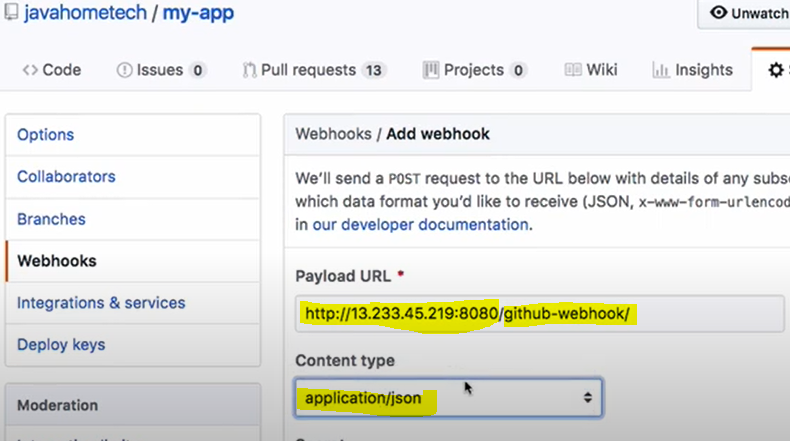
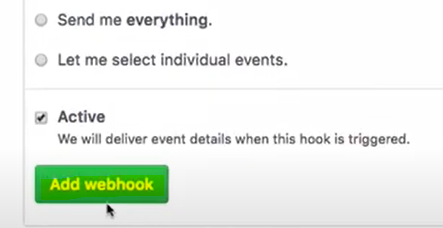
git rebase master – it will combine c1, c2, c3 commit into single history into c5 commit history.

##################################################################################

##################################################################################

Git Hooks

Jenkins URL and need to mention “/github-webhook/”



**Reference:**

<https://youtu.be/PhxZamqYJws>