1. mkdir foldername = To Create a folder [mkdir = make directory or make folder]
2. touch foldername/filename{1..5}.txt = it will create 5 text files in that folder. {filename1, filename2, filename3, filename4, filename5}
3. cd folder name = To move from one folder to another folder
4. cd = Change directory or change folder
5. ls = it will show list of files and directories
6. cd .. = it will exist or move out from the current directory or folder
7. cd ../../../.. = it will exist or move out from the current directory or folder
8. git clone repositoryurl = it will download the repository

Note = For every repository there is a default branch called main

1. ls -al = it will show list of files and directories along with hidden files [ al = all long list]
2. touch filename = it will create a file
3. git status = to check whether the files are in the staging area or not
4. git add filename = it move your file into staging area
5. git commit -m”message” = it will commit your changes to local repository with message

To push your file into repository first you have to login into your git account in git bash by using commands

1. git config --global user.name “ your username” we should our username and email in double quotes. Doublle quotes is mandatory

**Example : git config –global user.name “VenkateshD2510”**

1. git config --global user.email “ your email”

**Example : git config –global user.name “Venkatesh.dasari66@gmail.com”**

After login only you have to commit the file. After commiting you have to push the file

1. git push = It will push your changes to remote repository
2. cat filename = to print the content of the file
3. git pull = It will get the latest changes from remote repository to local system.

Git pull is a combination of fetch + merge

1. git fetch = it will track or download the remote changes but it doesn’t reflect your changes to your local system. To reflect your changes, you need to do git merge
2. rm -rf foldername = to delete the folders
3. rm filename = to delete the file
4. git init = it will initialize your normal folder as git repository [ git folder]
5. cat .git/config = it will show the content of the .git folder
6. git branch branchname = it will create a branch
7. git branch = it will show list of the branches
8. git branch -r = it will give list of remote branches
9. git branch -a = it will list of remote and local branches
10. git checkout branchname = to switch from one branch to another branch
11. git log = it will show the list of commit id’s [ for every commit it will create the ID’s]
12. git checkout -b branchname = it will create a new branch and switch to that branch
13. git add . && git commit –m”testcommit” && git push = By using && we can use multiple commands in single line
14. git merge branchname = to merge one branch into another branch
15. git cherry-pick commitId = Moving particular commit from one branch to another branch
16. git branch -D branchname = To delete branch in local
17. git push --delete Origin branchname = To delete branch in remote
18. git stash = It will move your files to stashing area with latest commit message and commit id
19. git stash -u = It will move your files along with untracked files to stashing area with latest commit message and commit id
20. git stash pop = It will get your changes from stashing area to local working directory and it will delete the stash list
21. git stash list = it will give list of stash entries
22. git stash save”message” = it will move your files to stashing area with custom message
23. git stash apply = it will get your changes from stashing area to local working directory but it wont delete the stash entry
24. git stash apply stash@{id} = it will get your particular stash id changes from stashing area to local working directory but it wont delete the stash entry
25. git stash drop stashid = It will delete the particular stash
26. git stash clear = it will delete all the stash entries
27. git stash branch branchname
28. fork = grtting public repository from one git account to another git account.. the fork option will be there in git account. Its not a command
29. git revert commitid = It will delete the particular commit
30. git reset --hard origin/branchname = It will reset your local changes upto latest pull
31. Creating Tags = A Lightweight tag is very much like a branch that doesn’t change. It’s just a pointer to a specific commit

**YOUTUBE GIT HUB TUTORIAL TELUGU 6:28:04**

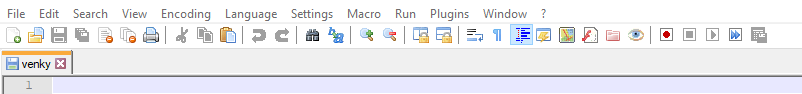
**URL = https://www.youtube.com/watch?v=LIhE7L\_\_E6M**

1. git config --global = it will show all our personal information like user name , email

[git config --global -e = it will show all our personal information like user name , email and editable}

1. :q = it will exist out of the command
2. clear = it will clear all the data in command prompt
3. git config --list = it will show all the list and data of the git config with our username and email that we have configured to github
4. git help config = it will show all the details of the config command . if you want to know all the details of command [ git <verb> --help]
5. notepad++ filename = it will open notepad++ and created the with the name of filename

**example = notepad++ venky = it will create with the name of venky**



1. git config --global core.editor “notepad++.exe -multiInst -nosession” = to make a notepad++ as a default editor **[For mac we can use TextMate in place of notepad++] 1:38**

CLONING

**Cloning =** cloning means downloading. By using clone command and repo url we can download the repository into our local system from git hub server. We can use http url or ssh url. By using http url we cannot get issues.

1. git clone repository url = it will download the repository into our local system from remote server (github server)
2. git clone repositoryurl prename = it will download the repository with the name of prename . (we can give any name in place of prename)

**INITIALIZING GIT REPOSITORY IN EXISTING PROJECT**

The Process to add our local folder to repository

1. git init 🡪 it will convert normal folder to git(master) folder. After using this this command .git folder(hidden folder) will be created.
2. git add -🡪 The folder or files will go to staging area
3. git commit -🡪 it will be commited
4. git remote add origin repourl -🡪 the folders will add to a particular remote and particular repository by using this command. So we need the particular repository url. In place of origin we can give any predefined name. By using this we link the server to our local system
5. git push -u origin master -🡪 by using this command we push the project ( our folder) to repository

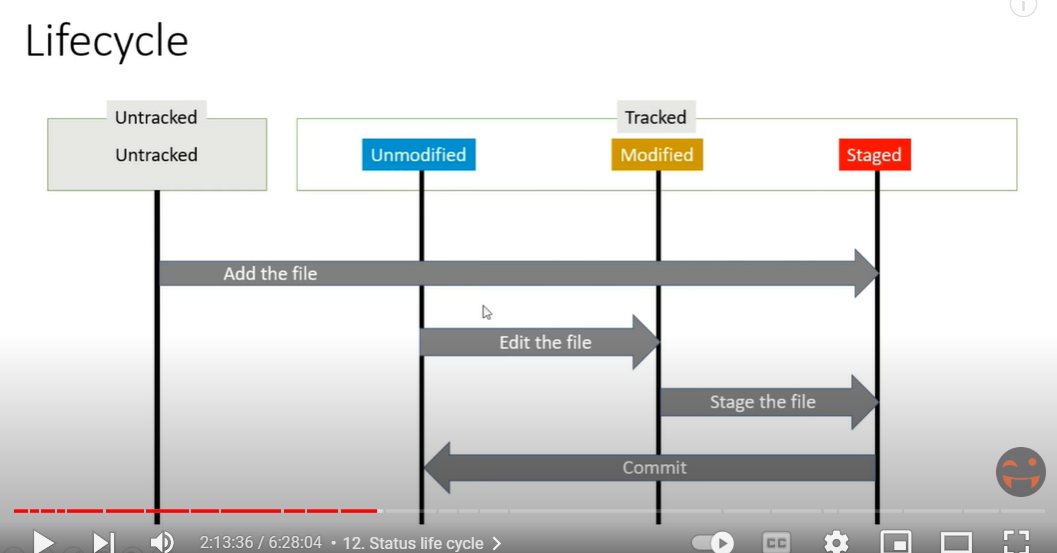
master means we are pushing the master folder

**STATUS LIFE CYCLE**

In git we have untracked files and tracked files.

Files which don’t know to git are called untracked files and files which know to git are called tracked files.

**Tracked files =** In tracked files we have Unmodified, Modified and Staged.



1. git status –short [git status -s]
2. ?? -🡪 Untracked Files
3. A -🡪 Staging Area
4. M 🡪 Modified Files

**IGNORING FILES**

If you want to ignore a file that you've committed in the past, you'll need to **delete the file from your repository and then add a .** **gitignore rule for it**. Using the --cached option with git rm means that the file will be deleted from your repository, but will remain in your working directory as an ignored file.

#comments can be placed

test.txt

test1.txt

\*.txt

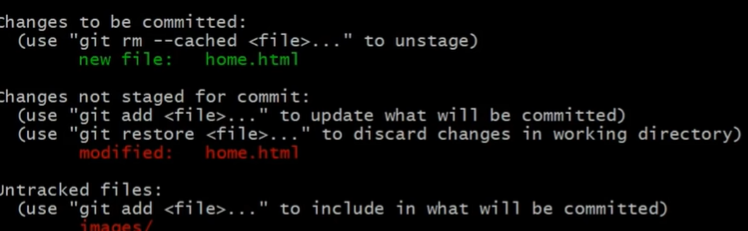
build/

2:36:27

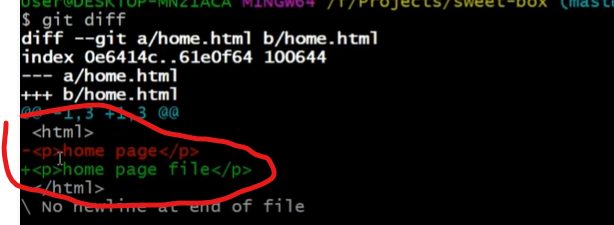
**GIT DIFF**

**STATUS VS DIFF**

Git status will show the file .If we change or modify the content git status will show the modified file

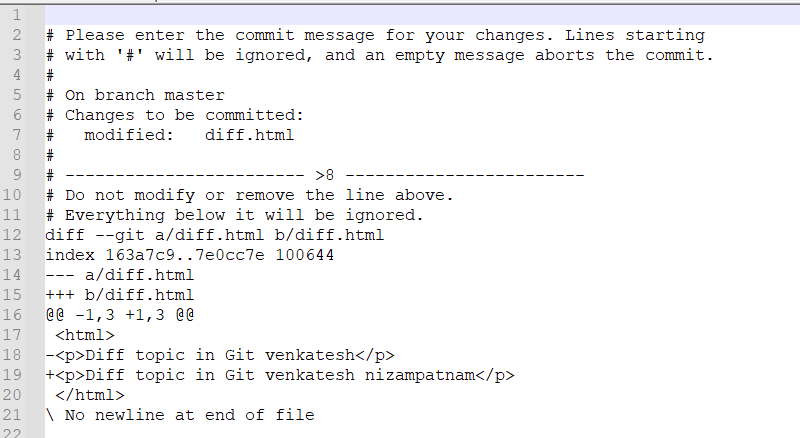


git diff will show the modified content(data) of the file .After Staged state if we modify the file then it will go to modified state. If we want to see the content of the modified file then we have to use git diff command.



**COMMIT**

1. git commit -v = It will show the modified data in commit message(notepad ++) [here -v means verboge} if we want to know the what data we have modified then we can use this commit

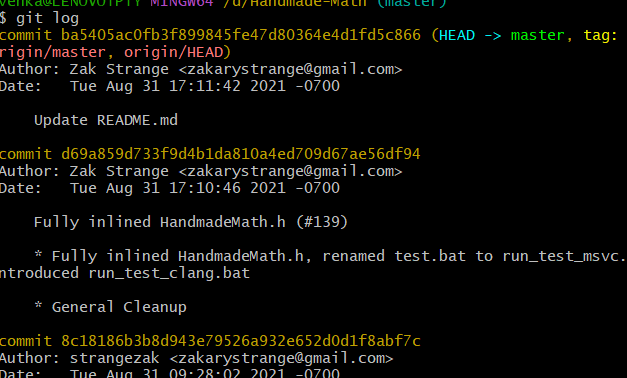


1. git commit -m “inlinecommit” = This is Inline commit [This command is called inline commit. Mostly we use this commit]
2. git commit -am “message” = this command is for to add and give commit at a time

**LOG HISTORY**

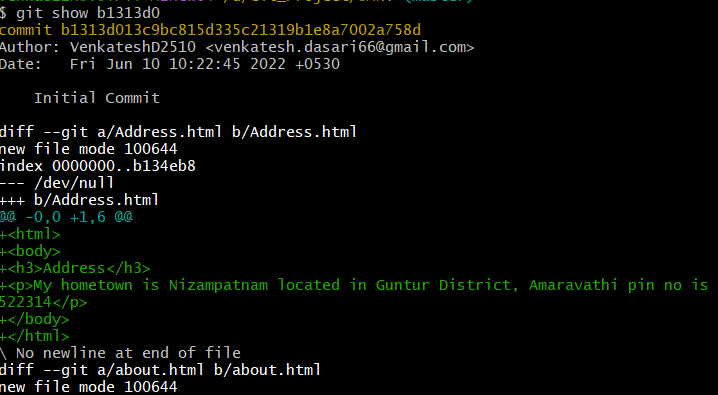
If We want to work the existing project. To know the history of the project then we can use the log history, and we can know the last 10 days work.

1. git log = It will show the previous commits



1. git log -p = It will show the latest commits along with modified data and commit ID’s
2. git log -2 = it will show only 2 commits
3. git log –pretty=oneline
4. git log –pretty= short
5. git log –pretty= fuller
6. git log –oneline –all -🡪 It shows all branches commit information. Not only one branch information.
7. git log –oneline –graph –all
8. git show commitid = it will show all the modified data for that commit

Example = git show b1313d0



**BRANCHING**

**MASTER**

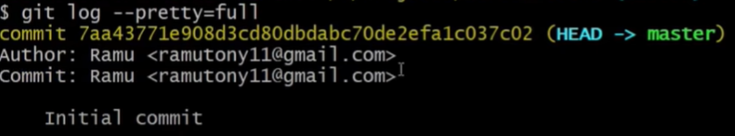
After initializing folder to git that folder becomes as a master. Normal folder should not be a master folder.

Downloading repositories to our local system that repositories(folder) are also becomes a master.

**COMMIT**

When we are doing first commit then it is called as root commit .

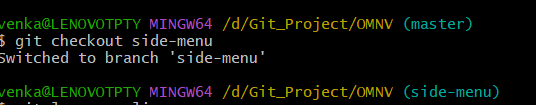
Commit Id is in 40 characters(digits+characters). Every commit is stored in a snapshot

Author = Who creates the repository

Commit = who worked on the issues and who committed the commit.

**Branch**

git checkout branchname = to switch from one branch to another branch

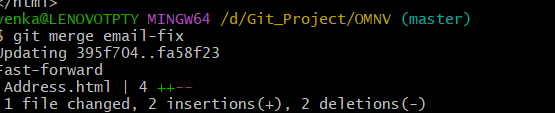


git checkout -b branchname = it will create a new branch and switch to that branch

git merge branchname = to merge one branch into another branch. if you get issues in project then you have to create separate branch for that issue and after solving the issue you have to merge this issue branch in master branch

To which branch you have to merge first you have to checkout to that branch.

For example, I want to merge email-fix branch to master branch, then first I have to check to master branch then merge email-fix branch to master branch



After solving the problem you can delete the solved branch (email-fix branch)

git branch -D branchname = To delete branch in local

**MERGE CONFLICTS**

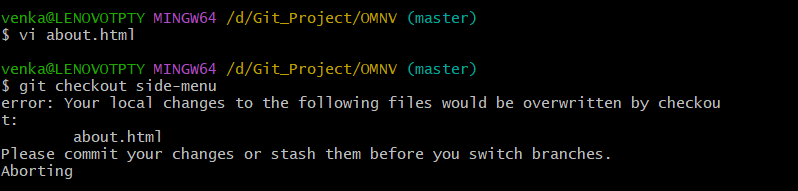
We will get more issues while doing merging

git branch –merge = by this command we can know what branches are merged

git branch –no-merge = which branches are not merged

For example two persons are working on same file and changed some code then we get merge conflicts

If we change any file in one branch, after changing the file we mandatory commit the file in that branch before moving to another branch otherwise we get error below is the screenshot



4:24

**ALIAS , REMOTE BRANCHING , STASHING , REDBASE , TAGGING**

**REMOTE BRANCHING**

Here we use fetch, merge, pull and push topics

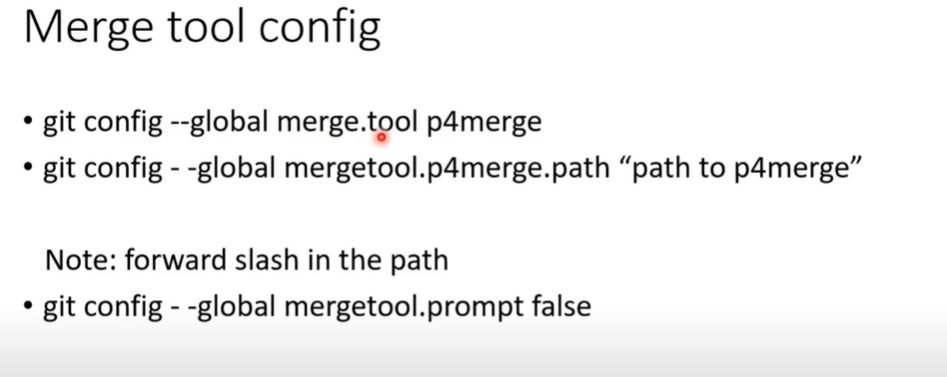
If we want to push our latest project to remote, first we have to do pull because if any changes are there in remote then we have to take updated project to local system. After doing pull only we have to do Push Operation

In real scenario we didn’t push projects directly to remote.

First we push to dev branch if there is no issues then we push to prod branch

**P4MERGE TOOL**

We can rectify issues pull issues by using P4 Merge tool



**ALIASES**

git log –oneline –graph –all -🡪 we can convert it as alias to allcommits. To convert it as alias we have to use below command

git config –global alias.allcommits “log –oneline –graph –all” = After executing this command below git allcommits command will work and show the commits

git allcommits

**REBASE**



Rebasing is a process to reapply commits on top of another base trip. It is used to apply a sequence of commits from distinct branches into a final commit. It is an alternative of git merge command. It is a linear process of merging.

**STASHING**

Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work. Git stashing allows you to do so. The **git stash command** enables you to switch branches without committing the current branch.

Generally, the stash's meaning is "**store something safely in a hidden place**." The sense in Git is also the same for stash; Git temporarily saves your data safely without committing.

**CLEANING**

git clean -f = it will delete untracked (newly created files) files .it delete only untracked files not folders

git clean -f -d = it will delete untracked folders

git clean -f -d -x = it will delete untracked files and .gitignore files

git clean -f –dry-run / git clean -f -n = it will show which files will be remove by that command

**TAGGING**

Tags have two types

1)Lite weight tags = a tag with only name. it don’t have any description

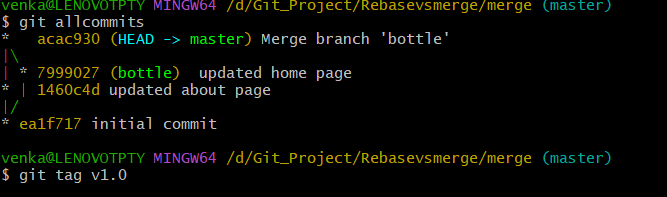
2)Annotated Tags = A tag with name and description

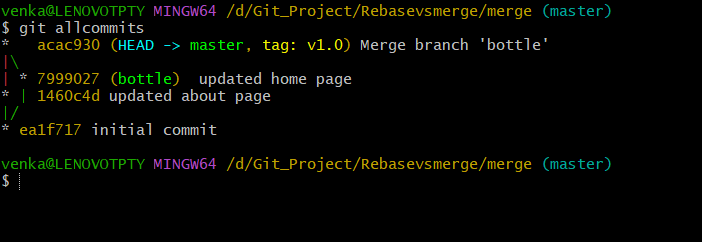
Tag means we are marking one particular commit to a tag. We are creating Tag for one particular commit

Example we have a lot of commits in project. After 2,3 years We don’t know at which stage of commit we release project for testing, production. For testing stage we can create one Tag like v.1(version 1) or any other name. and for production stage we can create second tag (v.2)

git tag --list / git tag -l / git tag = It will show the existing tags

git tag tagname = To create a tag (Example = git tag v1.0) Tag will create for latest commit. These tags are calle lite weight tags





git tag -a tagname -m “description “ = These are the annotated tags

git diff v1.0 v2.0 = To compare the tags

git tag -d tagname = to delete the particular tag

git tag -a tagname commitid = it will create tag for particular commitid (example = git tag -a v1.0 ea1f717)

git tag -a v1.0 -f commitid = to change the existing tag to another commit

git push origin branchname –tags = it will push all tags from local directory to remote

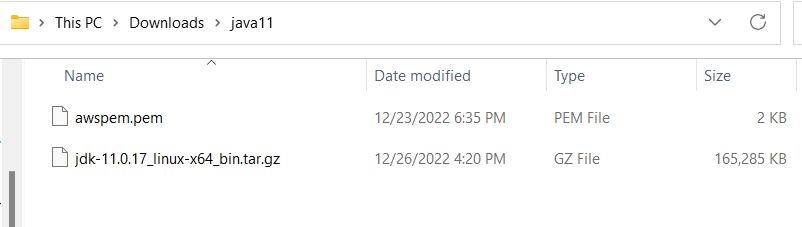
git push origin master v1.0 = it will push v1.0 tag from local directory to remote

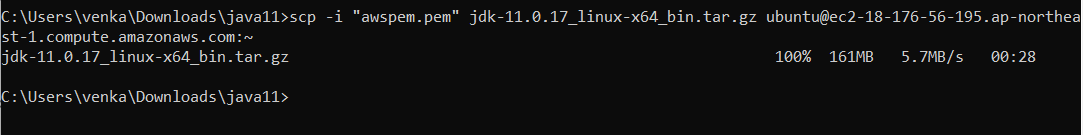
git tag -list “v1.\*” = it will shows all tags with the name of v1 like v1.0,v1.2,v1.3 etc….

scp -i "awspem.pem" jdk-11.0.17\_linux-x64\_bin.tar.gz [ubuntu@ec2-18-176-56-195.ap-northeast-1.compute.amazonaws.com:~](mailto:ubuntu@ec2-18-176-56-195.ap-northeast-1.compute.amazonaws.com:~) = to copy file from windows to linux

jdk-11.0.17\_linux-x64\_bin.tar.gz = this is the file name

To copy we have to open cmd in local machine (windows) and navigate to where the pem key and copy file are available





Jdk alternatives

<https://www.guru99.com/how-to-install-java-on-ubuntu.html>

Copy-paste below lines into a command prompt.

sudo update-alternatives --install "/usr/bin/java" "java" "<Directory where JAVA has been extracted>/bin/java" 1

sudo update-alternatives --install "/usr/bin/javac" "javac" "<Directory where JAVA has been extracted>/bin/javac" 1

sudo update-alternatives --install "/usr/bin/javaws" "javaws" "<Directory where JAVA has been extracted>/bin/javaws" 1

sudo update-alternatives --set java <Directory where JAVA has been extracted>/bin/java

sudo update-alternatives --set javac <Directory where JAVA has been extracted>/bin/javac

sudo update-alternatives --set javaws <Directory where JAVA has been extracted>/bin/javaws

du -h filename = it will show size of the file