**To reduce the clone time :**

Git clone <URL> --single-branch <branch name>

Git clone <URL> --depth 1 🡪 only latest commit is considered

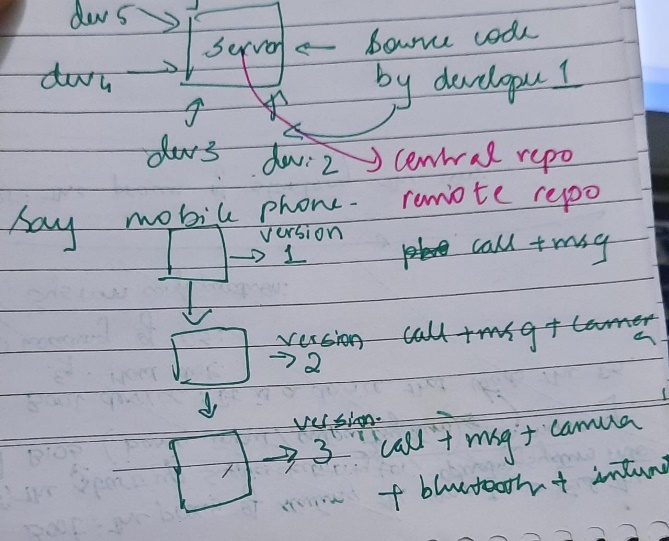
**ghp\_UIEokysU3MLq8jCPJwm9qAjGkxNE0A1Y4Qws**

**Class 15 : 04-11-2022**

Teams in a project are development team, testing team and Devops

The source code developed by development team has to be converted to binary for machine to understand.

So once the development team completes the assignment they store the human readable format of the code it in a separate common server called central repo or remote repo.



GIT --> is a version controller tool used to keep track of versions of files and directories i.e all logs of files and directories.. We use version control tool to manage the source code.

For each file we can see the history of the file, who modified, who created, date etc.

Every details about each version will be stored in version control tools.

Most famous version control is GIT. Other version control software are

1. PVCS
2. Perforce
3. Apache subversion
4. GNU bazaar and many more

Git is 3 tree architecture

*Git add git commit : Versions will be tracked*

WORKSPACE ------------------> INDEX OR STAGING AREA ----------------------------> GIT REPO

Files in workspace area and Index area will be visible in all the branches.

When u r staging area v can unstage it and edit it in workspace, it wont be tracked.

We can also edit in git repo, only thing is new commit/versions will be created.

1. Do ls -a in an empty folder. U see nothing.
2. Sudo yum install GIT
3. **git init** ---> create git repo
4. (ls -a and make sure .git repo is present)
5. **git config --global user.name "abc"** ---> to add users.Only need to create once.

**git config --global user.email** [**abc@gmail.com**](mailto:abc@gmail.com)

**git config --global**

By default it takes “cloud user” as creator of GIT file.

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

1. vi file1 -----> (add two lines )
2. **git status** ------> (untracked files : nothing added to commit but untracked files present (use "git add" to track )
3. **git add file1** --> move files from local repo to index area
4. git status ( On branch master - No commits yet)

Changes to be committed:

(use "**git rm --cached <file>**..." to unstage from index area to local repo)

1. **git commit -m "added file1"** ---> this will move files from index area to git repo.

*m –message*

1. After doing this, you may fix the identity used for this commit with:

**git commit --amend --reset-author** ----> to change the message typed during git commit –m . For me it only worked to HEAD commit. Later just do git add once u change the msg.

1. **git status** ---->(nothing to commit, working tree clean)
2. **git log** ---> shows history of repository – version,who created,when,the message u typed.

Note:

1. **git log -2** --> shows you latest two commits ; git log -3 latest 3 commits.
2. **git log filename** ---> shows you all commits on a file
3. **git log -2 filename** ---> latest two commits on a file
4. **git log** ----> Gives u history of old repository, (u can see all commits)
5. **git add \*** ---> to add all files in a directory to GIT.
6. Latest commit is called as HEAD
7. vi file1 ----> (add extra few lines to it)
8. git add file1
9. git commit -m "modified file1"
10. git log (u can see two commits)
11. **git checkout Prev-Commit-ID** ---> used to switch to previous commit or switch to branch or switch to tag
12. cat file1 ----> (check content of a file) ---> will show the first version content
13. git checkout master ----> (switch /to master)
14. git log ----> u can add one more file test and repeat above steps
15. To delete a file : After deleting u need to commit. Else it wont get deleted.
    1. **Git rm –f filename**
    2. Git add \*
    3. Git commit –m “ blah blah“
    4. Ls ---> file is deleted
    5. Git checkout previous commit ID ---> the file is not deleted here. It is only deleted in the new commit.

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

Note:

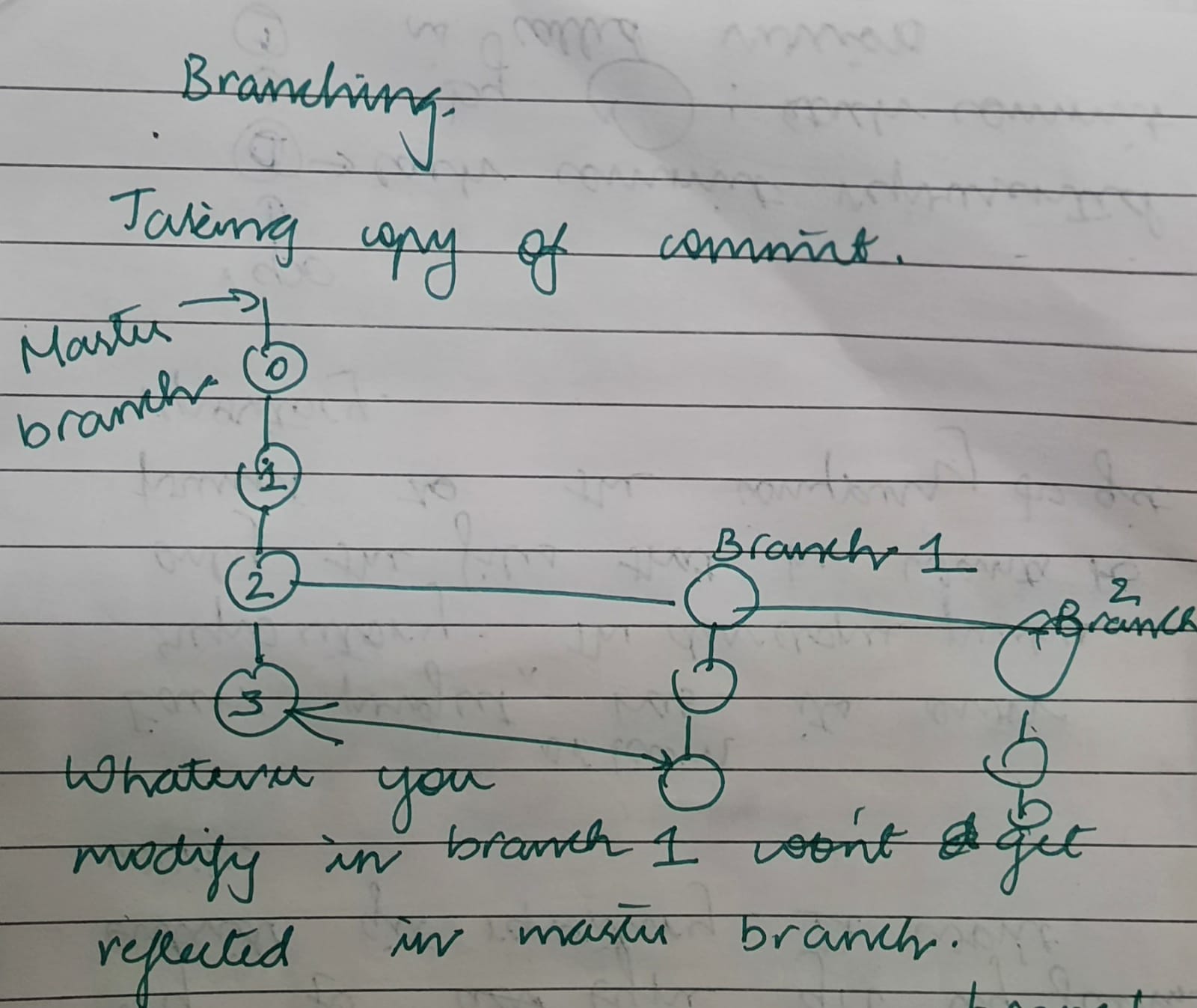
GIT works only in GIT repository. i.e .git should be present in that folder.

When you do git init , the folder will get converted to git repository.

Try to type git log filename outside git repository. It wont work.

Each team in a project has its own repository.

Branch ---> its for parallel development, two people or two team work on the same piece of code for developing different set of features.. Taking copy of the commit and integrate them by merging.



1. **git branch** ---> list branches. Default branch is master branch denoted with \*
2. **git branch branch1** ---> create branch from checked out branch
3. **git branch branch2 branch1** --> create branch2 from branch1.

Note: Branch created is not tracked in commit. I.e commit ID is not generated.

1. **git branch -d branchname** ---->how to delete branch
2. **git switch -c <new-branch-name>** ---> say u were doing “**git checkout commit ID** (commit ID of some old version of a file), when u do this it will ask if u want to create a new branch where commits created until this commit ID will be created as separate branch.
3. **git switch – or** undo the above operation with
4. **Git checkout –b branchname** -----> will create a new branch and checkout automatically.
5. **git checkout -b branch5 branch1** ---> create branch5 from branch1 and switch to new branch

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

Note: \* indicates which branch u r on.

Steps to practice

1. git branch branch1 ---> create branch1 from master
2. git checkout branch1 ---> switch to branch1
3. add new file in this branch , try switching user.
4. git checkout master --> go to master branch. The new file added in branch1 is not reflected here.
5. **git merge branch1** ---> branch1 will get merged to master
6. git checkout branch1
7. vi branch 1 ---> add more lines
8. git add file1
9. git commit –m “modified version”
10. git status
11. ls
12. git branch
13. git merge branch 1
14. create another branch branch2..
15. git checkout branch1 , git branch branch 2 branch 1 ---> new branch created from branch 1.

It will ask y u want to merge? Type the message. Now if u do git log, it will show this merging as HEAD commit. Also note that this changes in branch1 wont be reflected in master. If u want to reflect then u need to merge master and branch1.

-------------explain merge conflict---------------

1. Merging Conflict Will occur when the same peice of code is changed on 2 different branches, when we try to merge those two branches,then merging conflict will occur,
2. To resolve this issue, I don‘t know whose change should I take to merge , so I contact developers changes the code, person who modified code of branch1 and branch2. I get to know who modified the code on branch1 and branch2 using git log command.
3. Then they will decide and tell us whose changes should I take into merge.
4. Developers will raise new pull request as per the changes agreed upon.
5. Then I take that change and I commit it.
6. Once resolved the updated changes will only be reflected in branch1 and not in the branch2. The contents of branch2 will remain as such.

When u experience merge conflict :

Vi file ---> delete the contents that u don’t want

Git add \*

Git commit –m “ blah blah “

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

**Class 16 : 05-11-2022**

git tag---> tag is name given to set of versions of files. Its easy to remember in future and it indicates milestone of a project. Eg : release1

Tagging will enable the users to mark all the checkpoints in the project. When u want to add files to an old set of files , we can easily use that files by creating a branch from the tag.

Can tag the same files multiple times but everytime u need to give different tag name. Eg: release1.1

U only tag those versions of files which are working good and which u want to remember.

1. **git tag** --> list tags
2. **git tag tagname** --> create tag. tag the latest versions of all files (say release1)
3. ls
4. Git checkout master --->very imp step. Else all the files created further will be added to the previous tag.
5. Git checkout branch ---> where u hav created the tag
6. Now add one more file and commit it
7. **Git tag tagname2** --> Now new file is added and the version2 git tag is created (say release2)
8. **Git checkout tagname** ---> to log into particular git tab.

Note :

1. when u do git checkout branch above, it will show all the commits. Whereas when u do git checkout tagname, it will only show the tagged files.
2. Now when u do git branch , it will show git tag as the HEAD branch.
3. Git checkout branch3
4. Git checkout release2
5. Git log ---> will show the commit with tagname and branchname
6. Git tag ---> will show all the git tag created.
7. **git tag -d tagname** --> delete tag
8. how do you create branch from tag?
9. When u create a branch from tag, it will automatically create the branch from HEAD tag. Now say I want to create from older version of tag, then

git tag -a tagname commit ID -m "message" ----> If u want to tag different versions of

**git branch branch\_name tagname** ( tagname should be the version of that tag which u want to be created as a new branch” OR

**git checkout commit ID(of the one that u want to tag) 🡪 git tag tagname**

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

git stash :

if you are working on one branch, in between if you get any critical work which needs to be fixed on another branch.

in this case, before I switch to another branch, we need to stash it from current branch (stash will store it in temporary area).

Why is stash necessary? If u don’t stash file will be in WORKSPACE area. When u switch to another branch this file is available when u check git status. So chances are high that u might commit this file to the wrong branch and introduce a major bug

After completing work on another branch switch it to back to current branch and get stashed changes using git stash pop.

1. create some files
2. **git add \***
3. **git stash**
4. git checkout branch
5. after completing critical work another branch switch it back to current branch

git checkout current-branch

1. **git stash pop** ----> to bring it back to Index area. Can use git rm --cached <file> to unsatge the file, edit it and stage it back again.
2. git commit –m “ Blah blah “

**git stash list** ---> to get list of stash file

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

**git reset HEAD** ----> To undo the committed changes, but history will not be tracked. Its like deleting all the commit. This will reset only the HEAD.

Used to bring the stash to workspace

Types are soft, hard, mixed.

**Git reset - -hard commit ID** -----> Commit ID from which u want it to be head

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

**Git revert HEAD** ---> To undo the committed changes : Say u did not want to make the previous committed change. That time u use git revert HEAD, it will revert the latest commit and new commit ID got created.

Say u use git reset HEAD and only leave one commit. Now when u do git revert HEAD, it will undo this one commit ID also and leave no logs at all.

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

how do merge particular change or specific commit to branch??

If I merge branch 2 to branch1 all the changes of branch2 will be copied to branch1. If I only want to merge particular changes to branch1, then we use cherry pick

**git cherry-pick commitID**

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

Rebase :

Rebase can be used to club multiple commits.

1. git rebase ---> it's nothing but merge. one branch will get added to tip of another branch.
2. you can squash and merge it to another branch.

**Git rebase –i HEAD~4** ----> It means latest 4 commits u merge.

To retain the commit keep it as pick itself (Min 1 pick must be there)

To squash it change to s

**git rebase branchname**

wq!

**I have to always keep the first one as “pick” and squash the below commits. Else it will show error.**

**git rebase –edit-todo**

p, pick <commit> = use commit

# r, reword <commit> = use commit, but edit the commit message

# e, edit <commit> = use commit, but stop for amending

# s, squash <commit> = use commit, but meld into previous commit

# f, fixup <commit> = like "squash", but discard this commit's log message

# x, exec <command> = run command (the rest of the line) using shell

# b, break = stop here (continue rebase later with 'git rebase --continue')

# d, drop <commit> = remove commit

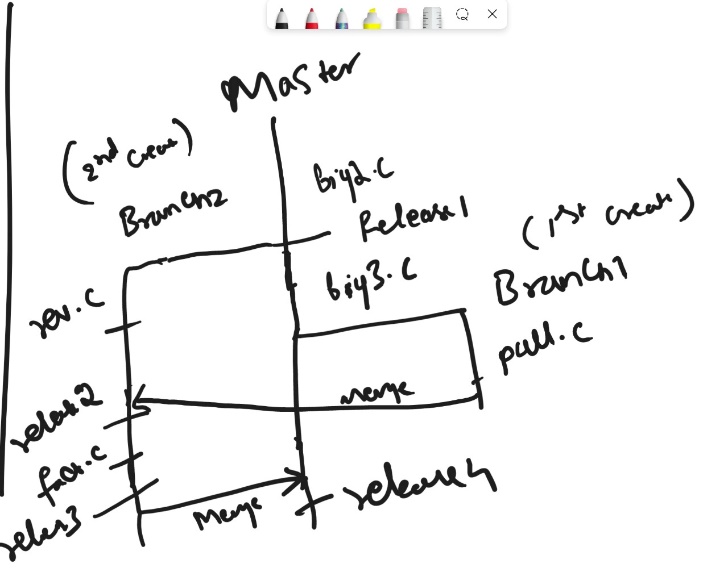
# l, label <label> = label current HEAD with a name

# t, reset <label> = reset HEAD to a label

# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]

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

Assignment :



=================================================================================

**Class 17 : 06-11-2022**

Git clone

Clone is bringing the copy of the code from central repository to workspace area. Git clone will copy all the data from central repo to local repo.

Bringing the remote repo to local workspace for the first time called as git clone.

Devops engineer is admin to git tool. Central repo is created by Devops engg.. We don’t store any machine language in git. We only store human understandable files in git.

Once a developer completes the work , he has to push to central repo, which is the confirmation that the developer has completed his task.

You can never open the central repo, u can only clone it.

The new joinee will clone the files from central repo. Later on the developer can only pull(Updates) the files from central repo.

1. Create folder named central\_repo in home
2. Create another 2 folders called dev1 and dev2 in home
3. Open central\_repo, **git init –bare 🡪** Creating central repo in server
4. Ls ----> .git wont be here, instead contents inside .git will be present now.

branches config description HEAD hooks info objects refs

1. Now open dev1 and dev2 and do the following
   1. **Git clone /home/ec2-user/central\_repo/** -----> If dev1 and central repo r in same server
   2. **Git clone user@server1:/home/ec2-user/central\_repo/**  ---> When u have to clone from server to another
   3. **Git clone URL\_of\_git\_hub**
   4. No need to use git clone. As when u clone the central repo the dev1 folder automatically became local repository.
2. How to clone a specific branch in git?
3. **Git clone –b <branch\_name> --single-branch <URL>**
4. When u clone the file and do ls, v can see .git.

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

Git push

Moving the changes from workspace to remote repo or central repo

1. Create a file – test1
2. Add + commit
3. **Git push /home/ec2-user/central\_repo**

**Git push user@severname: /path../central\_repo**

**Git push URL of remote repo**

1. Do this again

Git push user@servername: /path../central\_repo

Now when u tried 2nd time it will compare and copy only the files that is not there in central repo. So In this case nothing to be added.

1. **git push /home/ec2-user/central\_repo/ release1** ---> to push the tag created. Do this only after doing normal git push, this will be the second command. Or **git push --tag**
2. **git push /home/ec2-user/central\_repo : branch1** ---> to push the branch created. **Git push - -all origin**
3. **git push --set-upstream https://github.com/Manasabhat2222/backupjenkins.git master**

The current branch master has no upstream branch. To push the current branch and set the remote as upstream, use

Git pull

Bring the changes from remote repository and merges to local repo automatically. It will compare and pull only the changes file. This is the difference between git clone and git pull.

1. **Git pull user@servername: /home/./path/central\_repo** or **git pull /home/ec2-user/central\_repo**
2. **git fetch** ---> first in the master branch use this to pull the branch from central repo.
3. **Git checkout branchname** ---->will automatically pull the branch.

NOTE:

1. Add more lines to a existing file. Add + commit + push

When another developer pulls the files, then it will append the file and adds lines to file in local repo of this 2nd developer

1. Now in dev1 changed the content of a file2. (This is before u pull ). If u push to central repo and same file exits in central repo with the earlier content, then push will fail.

U need to first pull the files, then make edit, then push.

1. U pull make edit by changing the content, push the file. Say another developer will make changes in the file content before pulling it. Now when he tries to pull the file, it says merge conflict as same file has two different content. Resolve the issue of merge conflict and push the file.

NOTE :

1. Git , Bit bucket etc is latest version of version control tools which is a distributed version control system.
2. We can do the changes in both local repo as well as central repo.

When you pull code from central repo, then all versions of the file will be copied.

1. Perforce/SVN/CVS is centralised version control system which was used in olden days. Here only latest version will be copied.
2. U can do changes only in central repo and theres no concept of local repo.

**Fetch**

**git fetch** really only downloads new data from a remote repository and stores it in **remote branch** and - but it doesn't integrate any of this new data into your working files in **local branch**.

When u do not want to merge the data which you pulled from central repo to workspace, u use fetch. Do the necessary changes in remote branch and merge it with local repo.

1. First in git folder(where data is already cloned and stored) run the below command to list both remote branch (written in red colour) and local branch (written in green colour)

**Git branch -a**

1. Make changes in git hub repo
2. Git fetch 🡪 in terminal (it will show which remote branch the changes have been updated to
3. Git checkout <name of remote branch>
4. Make the necessary changes
5. Git checkout <local branch>
6. Git merge <name of remote branch>

Refer deeksith sn

Use case: If some one else has changed the same file which u were working on, if u want to check if changes are ok or not, then we do git fetch

**Git fetch user@servername:/home.//./central\_repo/**

Dev1

Create 2 files + add + merge

Push

Dev2

**Git fetch /home/ec2-user/central\_repo**

I am not able to push code to central repo. what is the reason?

other developer might have changed the same line of code in different way and push already. So first u need to pull the updated version of the file and make changes to that.

so if I am trying to push to central repo, I will not able to push.

in this case, I need to pull it and merge conflict will occur in the local workspace, resolve it,push it back to central repo.

you should pull source code to local repo and resolve merge conflicts if any. then push it to central repo

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

Two types of repo are:

bare repo: Eg : central repo

its central repo. we can push and pull code. we can't run any git operations here

**git init --bare**

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

non-bare: it's local repo where we can run all git operations.

when we clone central repo to workspace. it becomes local repo automatically. git init is present here.

**Git init**

***Additional information added by myself: How to fork ?/How to clone additional branch (not master) from git hub to local repo? Fork is done in the same way***

Note that if v use normal git clone command, it will not pull the sub branches. So use foll command

git clone –b <branch\_name> --single-branch <URL\_of\_the\_repo> 🡪 to clone the branch

git push origin <local branch\_name>:<remote branch\_name> (git push origin feature:feature)

Once u push, it will give u option of Pullrequest , u can send pull request from this feature branch to master branch.

**Hooks:**

Refer deekshith sn

We write scripts for imp events like commiting, merging, pushing etc.

Eg : Say if I have personal and professional folder. When we push from personal folder, I want the User.email and user.name to be set as [manasa@gmail.com](mailto:manasa@gmail.com) and manasa , in other scenarios it will be configured to work email and name i.e [manasawork@outlook.com](mailto:manasawork@outlook.com) and Manasabhat

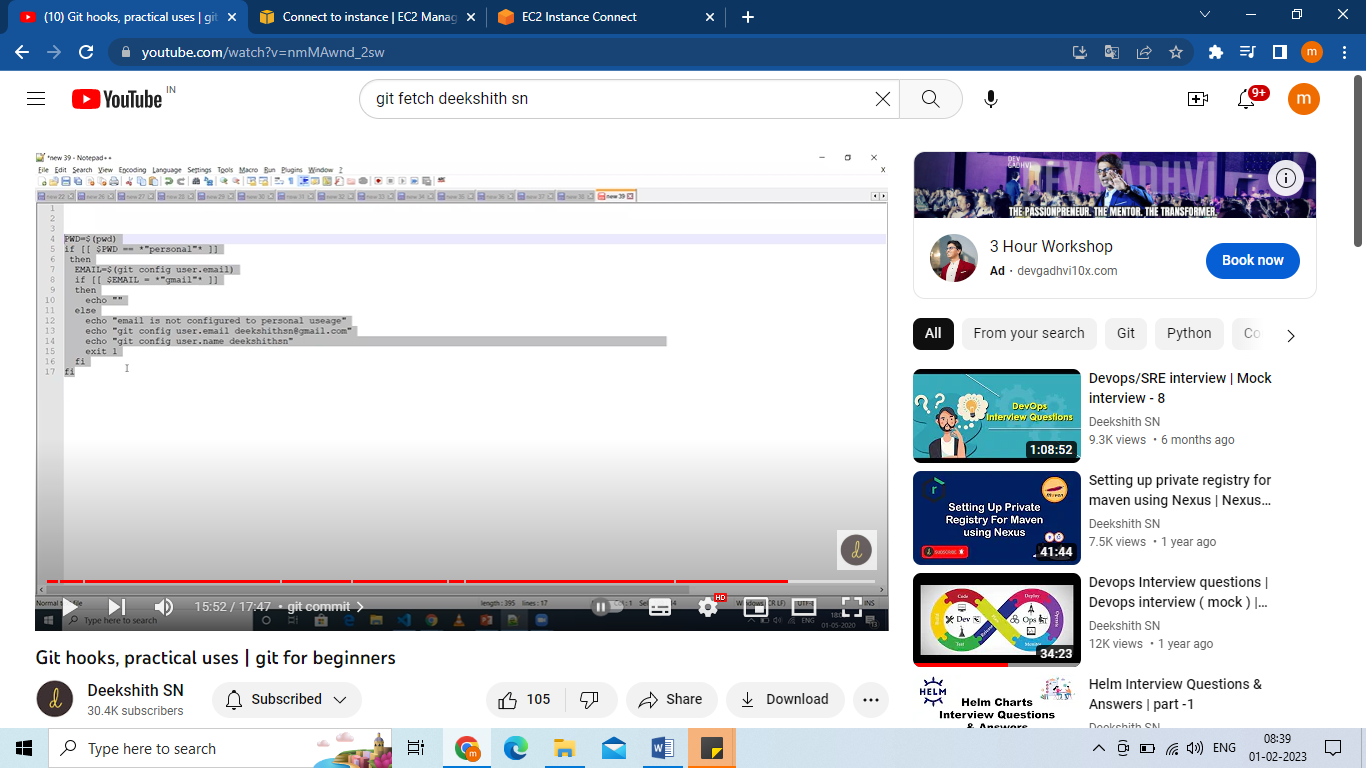
So when I use git commit “ “ command , then it should give me an error along with the message to set right email incase I use work email ID and name.

So to make this global and not local (not applicable to just one folder) I make changes in

/template/usr/git-core/templates/hooks

Mv pre-commit.sample pre-commit (std format, we need to rename it by removing sample)

Vi pre-commit



**Git commit “ “ --no-verify** 🡪 incase I don’t want git hooks to be applicable for this commit.

**Class 18 : 07-11-2022**

Alias : alias means same like.

Alias. man manasa means manasa is same like man.

Origin/master?

We need compiler to compile the source code into binary sent by multiple developers.

When we download a file/app in mobile we never use separate compiler to open the file/app. These files/apps are compiler independent. It does not need any tool, machine understands it as such without using compiler.

Compiler for .c : gcc

Comipler for .c++ : g++

Compiler for .java : javac

build tools

We need build tools to compile the source code.

Build : binary/executable/artifact

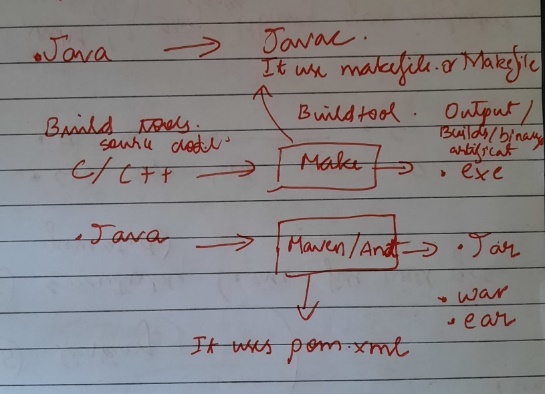
make is build tool used for C/C++. make looks for Makefile

Maven,gradle,npm and ANT is build tool for .java. Maven looks for pom.xml and ANT looks for build.xml

SUMMARY

.c 🡪 gcc 🡪.o 🡪 make 🡪 makefile 🡪.exe

.java 🡪javac 🡪 .class 🡪 maven 🡪 pom.xml 🡪 .war, .jar, .ear



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

STEPS OF COMPILATION:

1. Take the source code (.c files)
2. Preprocessing : Any library u included these in, it will check how much memory is needed for the variables and whether we have that many resources to convert into machine language.
3. Compilation
4. Assembly : Together 3) & 4) are called compilation. (.o files get generated)
5. Linking
6. Executable (.exe files are generated)

.exe files are dependent on .o files . .o files are dependent on .c files. This is what we write inside a makefile. We can generate a build only by using .o files also.

-------------makefile for two files f1.c, f2.c and f3.c---------------------------------

ABC.exe: f1.o f2.o f3.o

gcc -o ABC.exe f1.o f2.o f3.o

f1.o:f1.c

gcc -c f1.c

f2.o:f2.c

gcc -c f2.c

f3.o:f3.c

gcc -c f3.c

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

-----------syntax of makefile--------------

Generating target by using dependency files:

target: dep1 dep2...

<TAB> system commands

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

After saving main file and makefile file we type

make

It will generate .exe files and .o files which are in binary format.

We don’t need .o files, can delete it.

We give client .exe file called build. We never give the source code. Source code is secure with us. So if client needs any changes or features then he needs to pay additional charges. Only Chinese companies have the technology to convert these binary to source code.

Incase there is error in .exe files, then we need to edit the source code.

Check the size of du –sh\*. Size of .exe is more as it has 2 source code in it in binary format.

Usually there will be around 2000-3000 build files in a project.

BVT : Built verification test or sanity test

It is checking of basic functionality which is done by DEVOPS engg. It should never break. If it breaks then we inform the development team to check.

Eg: Phone’s basic functionality is to receive and make a phone call.

Once we check the BVT , we forward it to testing team to do thourough testing.

release ---> tested build which is ready to release to customer

------------------main.c--------------------------------

#include<stdio.h>

main() {

biggest();

factorial();

reverse();

}

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

---------------------big2.c--------------------------------

#include <stdio.h>

biggest()

{

int num1, num2;

// Ask user to enter the two numbers

printf("Please Enter Two different values\n");

// Read two numbers from the user

scanf("%d %d", &num1, &num2);

if(num1 > num2)

{

printf("%d is Largest\n", num1);

}

else if (num2 > num1)

{

printf("%d is Largest\n", num2);

}

else

{

printf("Both are Equal\n");

}

// return 0;

}

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

---------------------------------------------fact.c------------------------------------

#include <stdio.h>

factorial() {

int n, i;

unsigned long long fact = 1;

printf("Enter an integer: ");

scanf("%d", &n);

// shows error if the user enters a negative integer

if (n < 0)

printf("Error! Factorial of a negative number doesn't exist.");

else {

for (i = 1; i <= n; ++i) {

fact \*= i;

}

printf("Factorial of %d = %llu", n, fact);

}

// return 0;

}

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

----------------------------------------rev.c--------------------------------------------------

#include <stdio.h>

reverse()

{

char str[1000], rev[1000];

int i, j, count = 0;

printf("\nEnter string to reverse");

scanf("%s", str);

printf("\nString Before Reverse: %s", str);

//finding the length of the string

while (str[count] != '\0')

{

count++;

}

j = count - 1;

//reversing the string by swapping

for (i = 0; i < count; i++)

{

rev[i] = str[j];

j--;

}

printf("\nString After Reverse: %s", rev);

}

------------------------------------------makefile------------------------------------------------------

ABC.exe:main.o big2.o fact.o rev.o

gcc -o ABC.exe main.o big2.o fact.o rev.o

main.o:main.c

gcc -c main.c

big2.o:big2.c

gcc -c big2.c

fact.o:fact.c

gcc -c fact.c

rev.o:rev.c

gcc -c rev.c

clean:

rm -rf \*.o

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

GIT HUB : We never create central repo in linux server. We create in git hub.

Create git hub account

Email ID

Password

Username

Create repo : Give name : Create : Copy of the link : go to linux server : Create ur own workspace (a folder) :

1. Git clone *copy of the link*
2. cd filename
3. ls –a (.git will be created)
4. git push *copy of the link*
5. Username :
6. Password : Token
7. Git pull *copy of the link*

STEPS TO GENERATE TOKEN:

Go to git hub -> Setting -> developer settings -> personal access tokens -> token classic -> generate new token -> select scope : select all the items in linux -> generate token -> Token name : -> copy the token and save it somewhere.

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

Assignment1

include below features to ABC.exe

big2

fact

reverse

pallindrome

big3

sum of two numbers

fibanaci

sorting numbers

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

Assignment2:

create github account and create separate repo for c project

push all source code to this repo

(make sure you are not adding .o and .exe file to github repo)

git clone url on server

copy all .c and makefiles to this repo

git add

git commit

push it to repo

username :

password : (give token)

how to generate token

**Class 19: 09-11-2022**

DEVOPS engg are like cooks. They take all the source code and compile and build. We may know to use the tools, but we need to use right process, technique and right tools at right time to become successful DEVOPS engg.

The testing team accepts only the build from DEVOPS team. Coz chances are high that development team will include some local file and show that everything works fine. We on the other hand build it in our environment where there are no local files.

Central repo

Developer

Customer

Tester

DEVOPS

JIRA

Earlier we had two tools:

Bug tracking tools : Testing team will identify the bugs and assign it to development team.

Ticketing tool : Ticketing tools is used by most of them. From DEVOPS to developer, from tested to Developer to raise query.

JIRA

It is a ticketing tool. The one who raises the ticket is the owner of the ticket.

Status of ticket:

Open (When someone raise the ticket ) 🡪 Assigned( It is assigned for someone along with priority to resolve the ticket) 🡪 In progress ( The work is started , but not completed yet, can check the status of the work )🡪 resolved/fixed (Once resolved) 🡪 testing (He will check if the issue is resolved or not) 🡪 closed (Owner can close it/ the one who fixed the bug can also close provided he gets the permission from owner.

In Assigning stage we also need to mention the priority/criticality of the ticket. Priority is based on timeline.

Critical : Say within few hrs. Resolve the bug within specified time

Major : Say 72hrs

Minor :Say 2 weeks

We can also assign ticket to ourselves incase we feel that we might forget some work. Just as a reminder.

Check JIRA and check mail are the two things which engg does as soon as he reaches office.

Developer raises most of the ticket as major, we have the rights to change the priority.

If u don’t do minor work for a week then that work becomes critical.

open --> assigned --> in progress --> resolved/fixed---> testing (re-open if testing fails) --> closed

When does developer write when he raises the ticket to DEVOPS team?

* Description of the issue
* Can also share the logs that they recieved
* How to recreate the issue
* Assigned to: Devops team
* Criticality of the issue

What are stories,epics,themes and initiatives in git?

Stories: feature or functionality that a user wants to be able to accomplish. They are usually written from the perspective of the end-user and describe a specific need or goal.

Epics: Similar stories are grouped togethered to form epic so that they can be tracked and managed as a single unit. Epics are used to provide a high-level view of a project's goals and objectives.

Themes: Themes are high-level concepts or categories that can be used to organize epics and stories that help us to ensure that the project's goals are aligned with the overall business strategy.

Initiatives: represents a strategic business objective that can span multiple projects and teams. It is a way to align work with the overall business strategy and ensure that all projects are contributing to the same goals.

Jira, stories, epics, and themes can be represented as issues in the issue tracking system and can be assigned to teams or individuals for completion. They can also be prioritized, tracked, and reported on to provide visibility into the progress of the project.

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

Makefile works on time stamp basis.

1. When there are many source code, they write makefile like this.

ABC.exe %.o

%.o:%.c

Clean: rm –rf \*.o

This means everything is included. So don’t have to mention each file.

Only need to mention what is excluding in makefile.

1. When u make changes/updates in source code during/after the build, then only those files will be compiled. Run “make” once u r done with the changes.

Makefile will compare the target file with dependency’s time. If dependency’s time is later than target, then it will regenerate the target.

1. As DEVOPS engg when u pull files from central repo, and run make , it will recompile only the changes.

Two types of build are :

1. Full build/ Load build : When u pull the files from git repo, delete the old .exe and .o files and recompile all over again, it is called full build. We compile source code from scratch by deleting all intermediate files,so it takes more time to build
2. Patch build/hot fix : When u only compile the changed file,it is called as patch build/hot fix. Hot fix means urgent fix required by customer.

This patch code , is pulled, compiled, build, tested and review is given within 5min to developers.

JAVA:

Target and dependency are mentioned in pom.xml.

.c .java

|| ||

.o .class

|| ||

.exe .jar or .war or .ear

When we run maven, it looks for pom.xml

Any software that interacts with hardware, it is built in C language – Eg: elevator,phone

Any application u consider fb, banking, u open in browser and do it , They are Java projects.

Webapp server:

When I build .war file, we want to check the features of the build in environment, then for that purpose we use webapp server which is third party.(This acts like phone for the SIM to work).

Once u generate .war file u need to put it(deploy/deployment) to webapp server(tomcat), then that application will be available over the browser, we can test if the application is working or not.

Different webapp server are:

1. Apache tomcat
2. JBOSS
3. websphere

Assignment 1:

1. Google for java project with pom.xml
2. Download
3. cd place where pom.xml is present
4. mvn clean deploy (try deploy or install)
5. Create separate repo for java proj in github and push all above source code and pom.xml to this server.
6. Next pull it to ur server
7. Next compile the source code

Note:

1. download java project with pom.xml... build it (make sure u installed java and set JAVA\_HOME)

JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64

export JAVA\_HOME

MAVEN\_HOME=/opt/apache-maven-3.8.6

export MAVEN\_HOME

PATH=$JAVA\_HOME/bin:$MAVEN\_HOME/bin:$PATH

export PATH

Assignment 2:

1. Install tomcat on ec2 instance and deploy(access it over the browser) any war(calendar.war) file to web-apps folder. To deploy any build we need a webapp server.
2. Ipaddress:8080
3. Change tomcat to other available port

Very imp Step : How do u deploy .war file/Steps for deployment to a server on which apache tomcat is installed.

Say build1.war already exists. We need to deploy build1.1.war which has latest features.

1) stop tomcat services

2) take backup application logs 🡪 When u go to any website, any activity that u do will be stored in the form of logs which will be stored for sometime.

3) take backup of current build 🡪 build1.war. This is very imp coz incase the new feature doesn’t work properly, then we can reset to this build again.

4) copy new build 🡪 build1.1.war

5) start tomcat services

Its like switching off the phone to replace the SIM in mobile.

we deploy war file to tomcat server. we have shell script to deploy war file to tomcat automatically

script will do all those steps mentioned above

Assigment3:

learn what are maven targets and explain each goal

explain deployment process

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

**Class 20: 10-11-2022**

Development process :

1. Gather the requirement
2. Design
3. Build/coding/development
4. Testing
5. Release

Developers can login to only development server.

Whereas DEVOPS can login to any server, development server, testing server etc. We are like admin.

Development will tell which version of java u need to install.

We closely work with development team in a project.

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

When client comes with requirements, it is asses by the development team, scope --> scope , time..

They design it. Those design are shared with both development as well as testing team so that development team can start working on the design and testing team will get an idea how it should work

Any issues found, they raise the ticket, any work u need from anybody, u need to raise JIRA ticket.

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

DEVOPS engg support 3 environments by deploying to 3 servers. Mostly for JAVA. For C projects v directly install on the device.

Once deployment is done, testing teams work will begin, they test it different environments which are

1. QA or SIT : Quality assurance system integration testing: As soon as we deploy(they’ll give server name) first environment we deploy is QA/SIT(i.e webapps of testing server). We check whether all the features are working or not. Once the testing is okayed here, we DEVOPS engg will deploy the same build to next environment as per JIRA request from testing team.

Note : There will be multiple servers for each testing environment and testing team will ask to deploy in any of the server based on the availability of the server..and each server will have webapp application installed. If the tester wants to deploy directly to UAT or Prod, then he needs to raise ticket in JIRA.

1. UAT or Prepod : User acceptance testing or pre-production: (To deploy the build at different time, ul need managers approval) . There’s not much difference between this and Prod environment, so if it is working here, then it will work in Prod envi. Hence all minor bug fixes will be tested here. This is entirely different testing team.
2. Prod : Production : Needs CAB (change advisory board-senior ppl) permission to change the deploy time. It will affect the business. Its very sensitive environment. When v need urgent deployment testing team will request for meeting with CAB, in CAB meeting CAB members, testing team members, development team member, DEVOPS engg and the person who gave the project will also be present. Usually we deploy and test once a week in production environment. This is also different team to do testing.

The moment u deploy in Prod it means customers have already started using.

Note: Testing team will give say 3hrs for deployment to Prod server (it takes around 10min to deploy). If there are any issues during deployment, then we need to fix the bug within the deployment window. Else we need to roll back to previous build coz the company will bleed money

UAT and prod environments. We need manager's approval to deploy to QA and UAT.

but need CAB(Change advisory Board) to deploy to production.

DEVOPS is a method/process. Once u start working in that process it becomes a role.

These are environments that a company must have. Some companies may have more environments too.

Say I deployed a build in QA and later in UAT , it worked fine. Now I deployed in Prod and it is not working, What is the issue? Ans: Version mismatch of some tools.

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

if deployment fails, we will try to debug and fix issues within deployment window

if I am not able to fix that issue. I will rollback to previous build.

(rollback means deploying old build if new build is not working or deployment fails when deploy new build)

-----------command to start/stop/restart tomcat----------------------------------------

sudo service tomcat start

sudo service tomcat stop

sudo service tomcat restart

sudo service tomcat status

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

makefile --> works on timestamp basis. if target's time is less than it's dependencies time, make will regenerate target.

if dependency’s time is less than target's, make will not regenerate target.

How to push from script to remote repo without adding username and token?

Click on right top circle 🡪 settings 🡪 SSH and GPG keys 🡪 copy ur public key of the instance here 🡪 save 🡪 enter the repo that u want to clone 🡪 code : copy the **SSH code**

Go to ur server and the folder 🡪 git clone URL (SSH code) 🡪 go inside the directory 🡪 add the file 🡪 git remote add origin URL (SSH code) 🡪git add \* 🡪 git commit –m “ “ 🡪 git push origin master .

So for every instance u need to generate public SSH key add the key to github account.

So for MK question, inside the cloned folder, write this script.

#!/bin/bash

touch $1

whoami > $1

git add $1

git commit -m "adding script"

git push origin main

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

***Maven :***

Maven is a Project management tool for Java projects.

Libraries consists of dependencies.

***Java Project structure:***

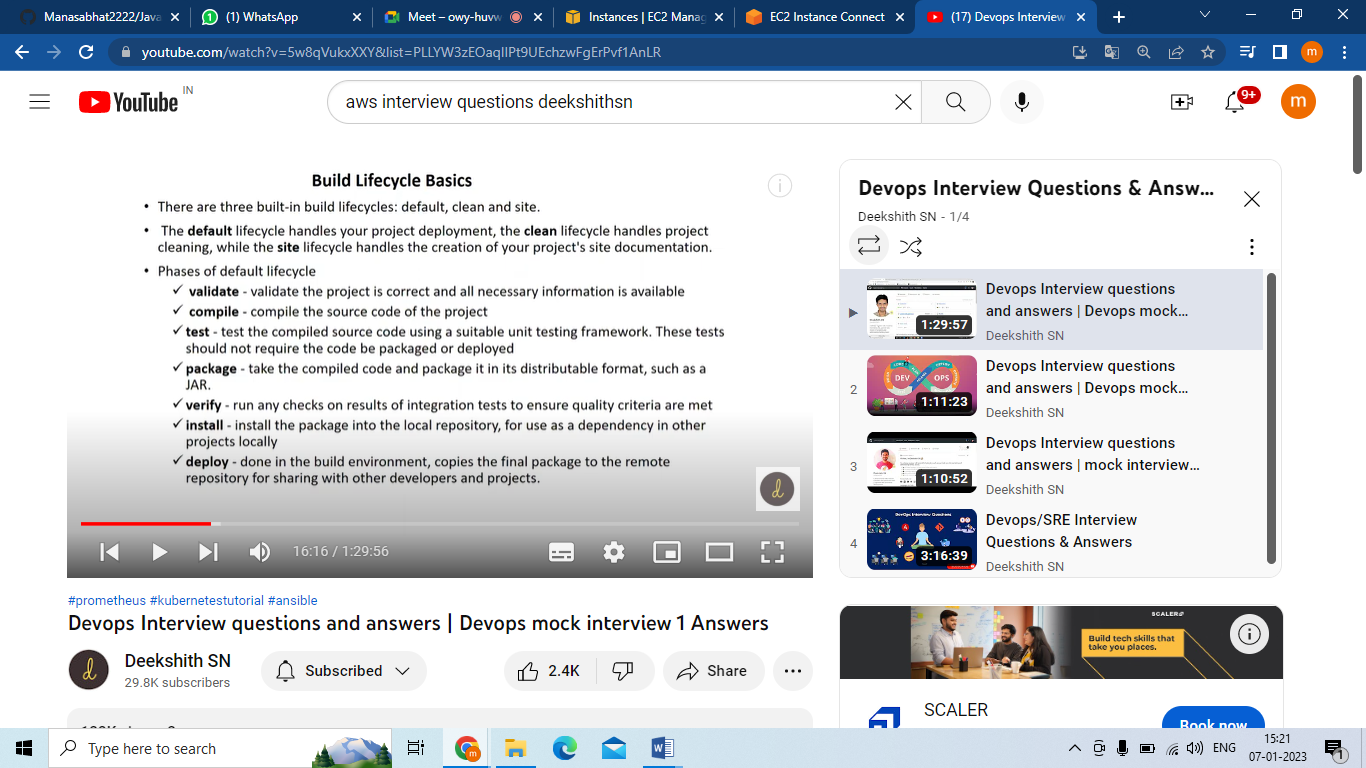
1. Source code (ur application code)
2. Test code (Test cases)
3. Project structure (assets, directories, resources) 🡪 it is a structure , eg: Inside abc folder -> xyz folder , inside xyz - .java. It should be in same standard format all the time.
4. Dependencies/Library
5. Configuration : which colour, font, alignment, text size, shape etc.
6. Task runner :Code to do build , test, run
7. Reporting

This is all managed by maven/gradle.

***Lifecycles of maven:***

1. Default/build : This has 7 phases explained below.
2. Clean: If u give mvn clear , it will clean the target folder.
3. Site : Creates documents about the project: no. of dependencies, no. of plugins, metadata etc.

***Maven Build Lifecycle:***



Validate: Validates the correctness of ur pom.xml, plugins, resources, source code etc ,whether all info has been provided properly.

Install: artifacts will be copied to .m2 which is nothing but local repo in maven.

***Settings that u need to have before mvn clean deploy:***

.m2 folder should have **settings.xml : id, URL of artifactory repo, username and password of ur artifactory repo. i.e**

**Mv settings.xml /.m2**

Pom.xml should have heading called distribution management under which 🡪 the settings.xml data and repo details in pom.xml should match.

Mvn deploy

***Why maven takes time to build the 1st time and it is faster the 2nd time?***

Since dependencies are downloaded from remote repo to local repo(.m2), the first time itself, so second time this step is avoided, hence making it faster.