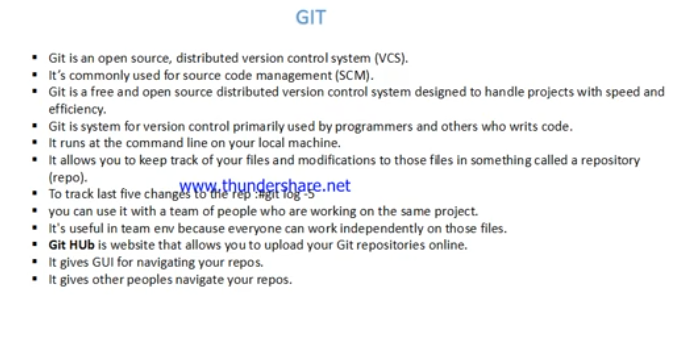
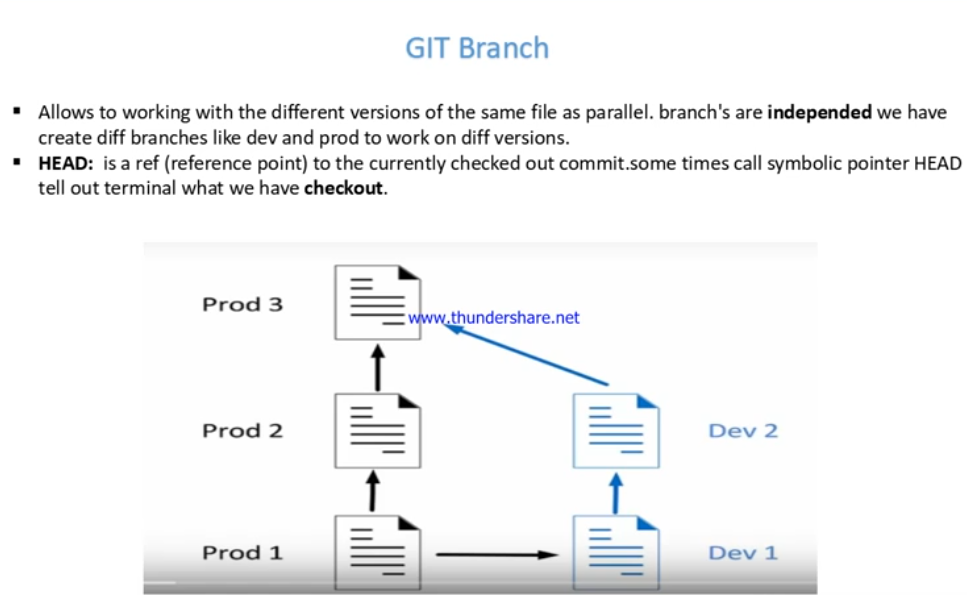
****

****

**New repository:**

**Create a new directory – # mkdir gkrepo -- # cd gkrepo**

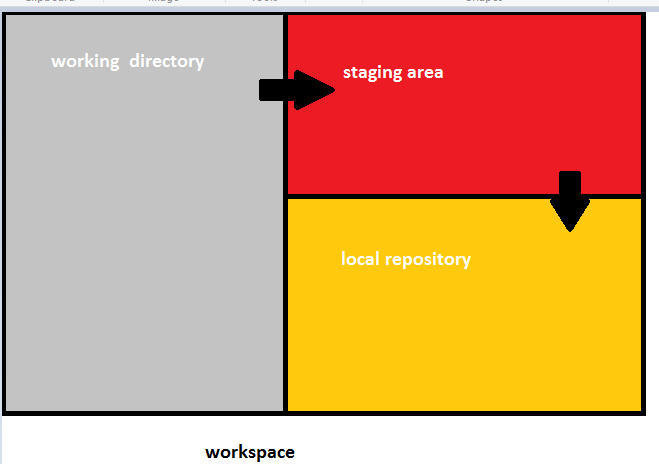
**# git init . :** ( here initialize empty repository) now “gkrepo” is act as repository.

**# ls -a :** one hidden directory is created **( .git )**

**.git contains complete branches related information**

****

**Interview quest) what is .git purpose?**

****

**# git init . :**  Here initialize empty repository

**# git add filename :** adding to staging area specific file only

# **git add :**  it means, at a time all files added to staging area

**# git commit -m “comment” :** add to local repository ( - m use for commit message )

**# git commit -a -m “comment” :** at a time add and commit

Each and every commit has generate one log

**# git commit --amend:** this option will modify the last commit message

Configure the author name and email address to be used with your commits

**# git config --global user.name “kadhar”**

**# git config --global user.email “vallikadhar@gmail.com”**

**# git status :** displays the state of the working directory and the staging area

**# git branch :** display list of branches. \* indicates your current branch

**# git log :** display all commits message history in your current branch

**# git log --branches=\*. :** display all commits messages in all branches

**# git log -Gword:**  Show the commits whose added or deleted lines contain word

**# git log --oneline :** Display the log file in single line

**# git show <commit-ID> :** display information about that specific commit

**# git branch branch-name :** create a new branch

**# git checkout branch-name :** Switched current branch to another branch

**# git checkout -b branch-name :** create new branch and switched that branch

**# git checkout commit-ID :** checkout a specific commit

**# git checkout --filename :** undo changes in your working directory

When create new branch from master or other. By default current branch data is sync to new created branch

**# git branch --list:**  display list of branches in your repository

**# git merge branch-name :** merge from current branch to another/given branch

**# git branch --merged master :**  list braches merged to master

**# git branch --merged** : list braches merged to HEAD (i.e. tip of current branch)

**# git branch --no-merged :** list branches that have not been merged

**# git branch -d branch-name** : delete branch

**# git branch -D branch-name** : force fully delete branch

**# git branch -a** : list both remote branches and local branches

**# git diff :** View all the merge conflicts

**# git diff :** Show changes between the working tree and the index

**# git diff branch1 branch2**

**# git diff commit1 commit2** compare two commits

**# git remote show origin :** If you want to see more information about a particular remote repository

**# git branch -v :**  to see last commit on each/all branches

# **git rm** : to remove files from both staging area and current working directory

# **git rm** **--cached** **filename**: remove files from the staging area and working directory

**# git clean :** The git clean command is used to remove all untracked files from your working directory ( untracked files means, files are not there in staging area & local repo, files in working directory only)

**# git clean -d :** remove untracked directories

**# git stash :** sometime we do not want to commit your code. But we do not want to lose the unfinished code. In this case we use git stash command.

It means whatever uncommitted code is there in your repository, all that unfinished work store in stash area. Now we can work in clean working directory.

**# git stash list**

**# git stash show**

**# git stash apply**

**# git stash pop:** remove stash

The latest stash you created is stored in refs/stash

**# git merge --squash dest-branch-name**

**# git checkout --filename :** undo changes in your working directory

**# git reset <file> :**  Remove file from the staging area.

**# git reset <commit ID> :**  back to specific commit ID

**# git reset HEAD~1 :** remove last(latest) commit ID your history

**# git reset HEAD~3 :** This will removes last 3 commit IDs in your history

**# git revert commit-ID :** So let's say you committed your code and then pushed it to the remote repository. undoing changes that were made in the specified commit by creating a new commit and not actually removing any previous commits

**# git rebase:** target branch commit ids will be get and keep it in top of your history and it will form a single straight line

Git rebase is integrate changes from one branch to another branch and kept in top of your history.

Git rebase is grab all the commits from particular branch and place them on top of the base branch.

**# git fetch:** retrieve/fetch all the changes made in the remote repository and store them in to local repository

**# git push origin master :** Pushing from Your local repository to centralize repository

**# git pull origin master :** get only updated/modified files from centralized repository. Simply downloading updated files only

**# git clone <…..remote repo..URL…>** : cloning an existing repository

**# git squash:**

Merge multiple commits in the same branch in to a single commit. It means we combine multiple commits in to a single commit.

Squash is used rewrite history.

**.gitignore:**  on any workspace. in which you want to ignore some files. Then create file in the name of .gitignore and inside this .gitignore file we put names of files. Whatever files you want to ignore then Git will be not consider those files

**# git cheery-pick commit-id**

**Git cheery-pick:**  you will have particular commit-ID that you have to merge to another branch

Pick the specific commit-ID that one is merge to target branch called cheery-pick

If someone wants to Commit specific commits in one branch to a target branch, then cherry-pick is used.

* Each repository has a 2 GB size limit, but we recommend keeping your repository no larger than 1 GB.

Delete a branch on your remote repository

**# git push origin :<branchname>**

Push the branch to your remote repository

**# git push origin <branchname>**

**git-archive :** Create an archive of files from

archive your local repository using git archive command in the format of tarball,zip

archive your all commits and paste in to specified location using git archive command

**# git archive -l**

Tar | tar.gz | zip | tgz

**# git archive --format=zip --output MyCommits.zip HEAD**

# ls

<zip file is appears here>

**git-bisect :** using git bisect to find bugs in your code

# get it ready

git bisect start

git bisect good c09c728

git bisect bad e6a0692

**Q) HOW CAN YOU RESOLVE MERGE CONFLICTS?**

Simple conflicts are We resolve manually and

If we have more conflicts then we use 3rd party tool like **diffmerge** and resolve the merge conflicts.

Download diffmerge tool and integrate with git

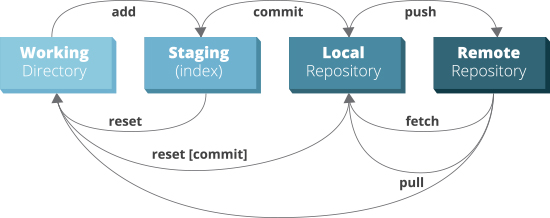
Multiple developers works on same file and same line in two different branches on that case we get merge conflicts

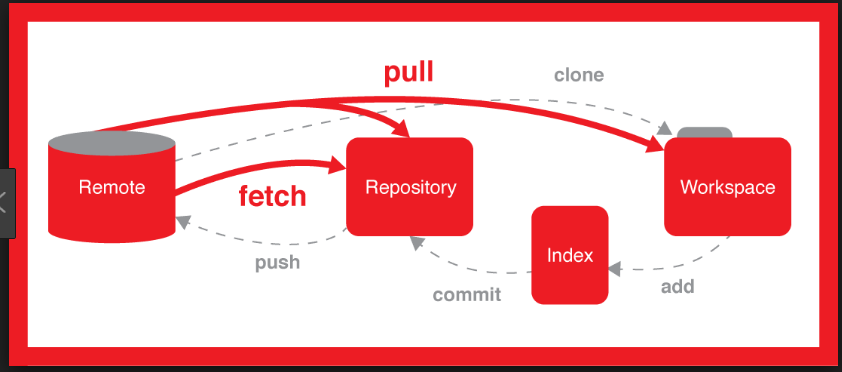
**List of merge tools :**  diffmerge, meld, vimdiff

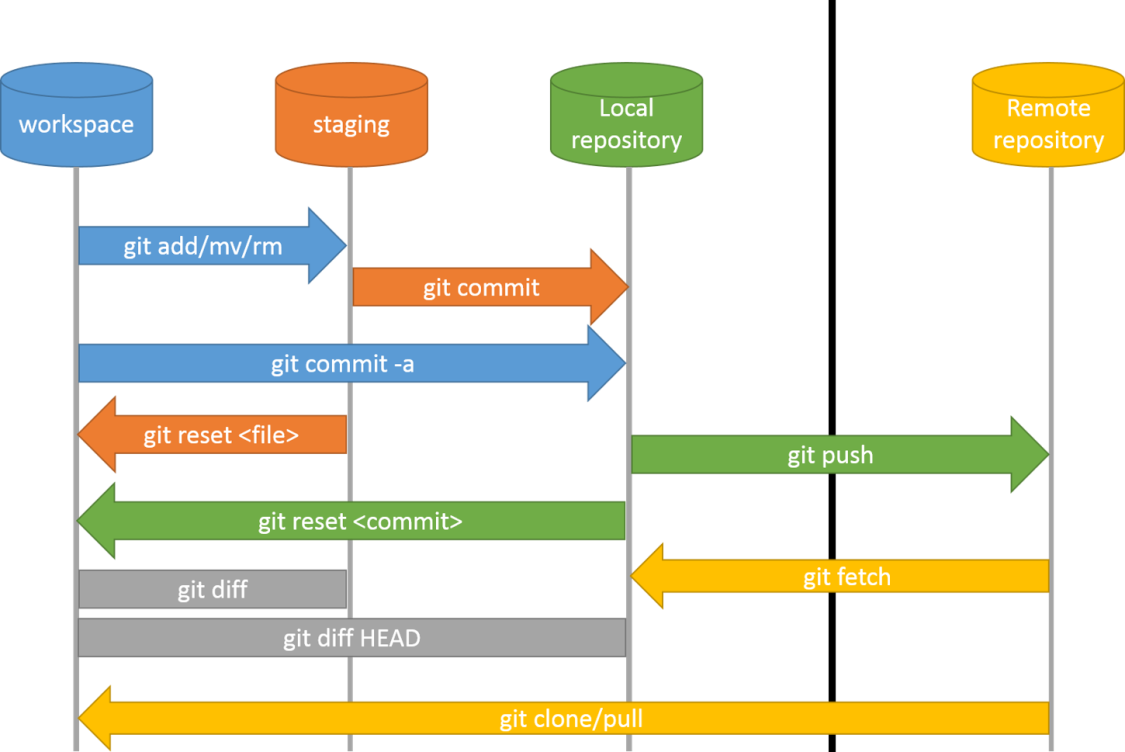
**Q) what is the origin in git?**

Origin is the remote repository in git. In which data flows to and from called origin.

Git origin contains details about remote repository

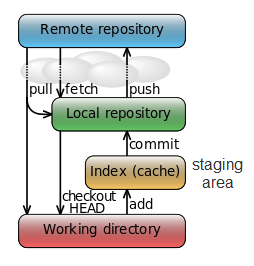


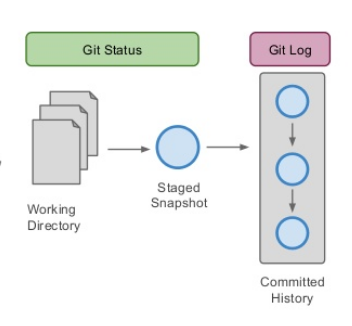


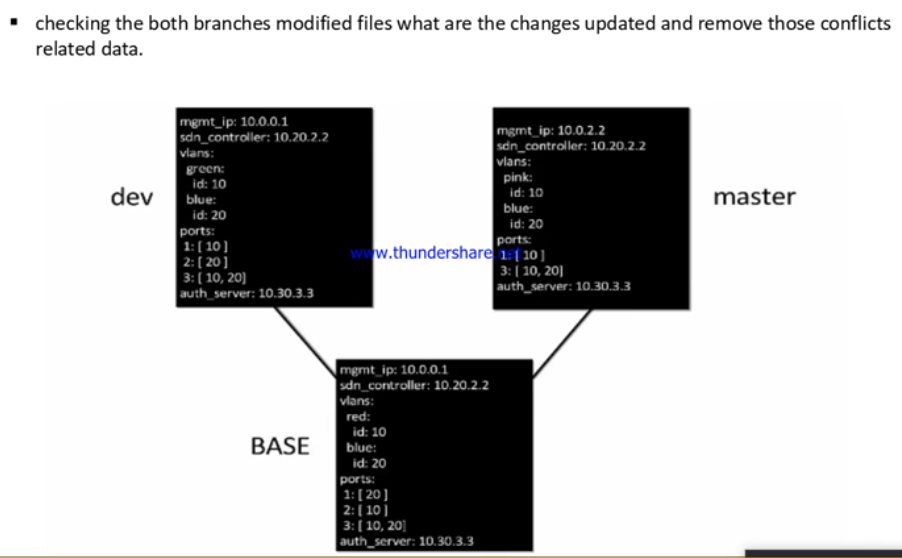


[**https://git-scm.com/book/en/v2**](https://git-scm.com/book/en/v2)

<https://www.atlassian.com/git>





****

**# git commit -a -m “comment”**

Git has two types of branches: local and remote

If you have another remote branches you have something like "git push origin test" then you pushyour changes to the test remote branch

**# git push origin master**

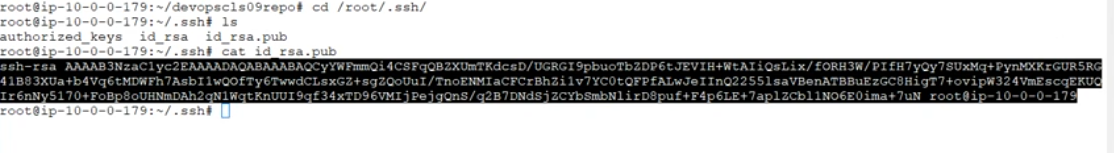
Here we Enter username and password of central repository. Every time enter credentials is time taken process.

Without asking credentials push directly using ssh-key

**# ssh-keygen:**

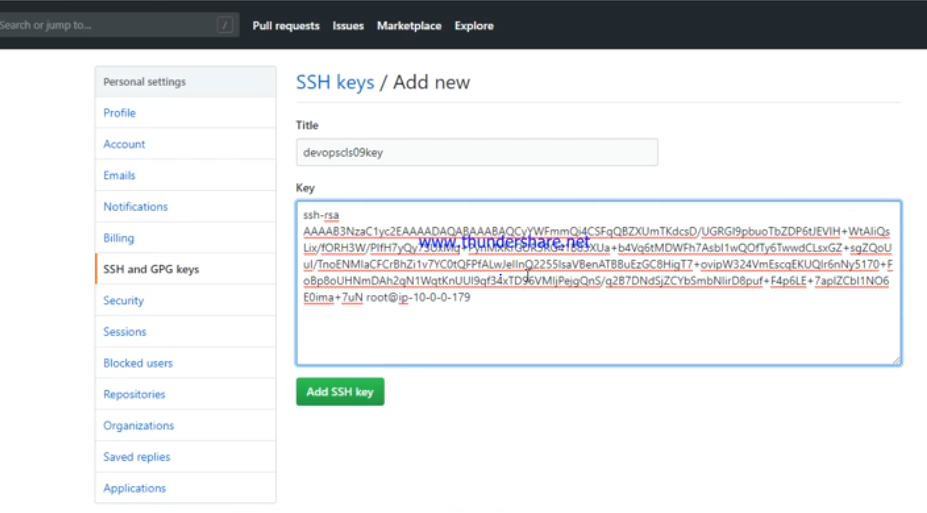
Press 4 times enter button

2 keys generated the location is # cd /home/username/.ssh or # cd /root/.ssh

****copy the public key : id\_rsa.pub

Go to git hub – go to account – select settings

Select SSH and GPG keys ---- create new key – give any name and paste the public key

****

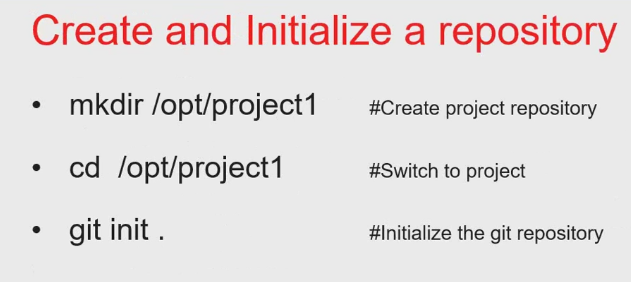
Click add SSH key and give password

**# git remote set-url origin < here paste git hub SSH path >**

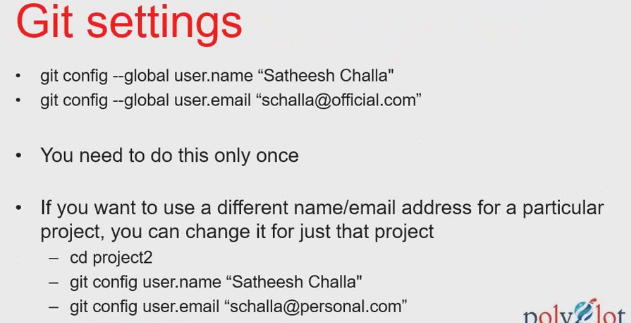
**# git push < here paste git hub SSH path >**

Using this way without credentials to push the code in to the centralized repository

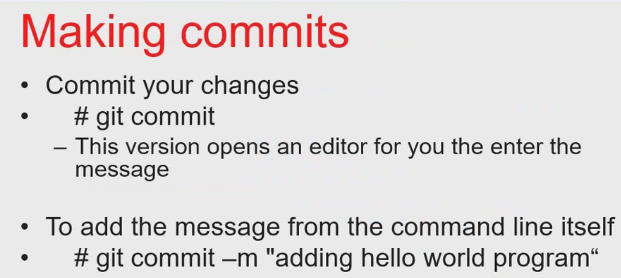
**SATHISH CHALLA @ polyglot**

****

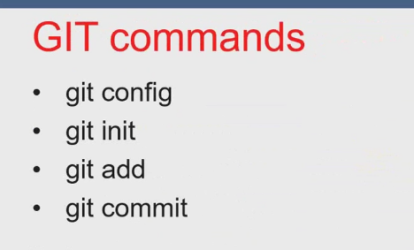
****

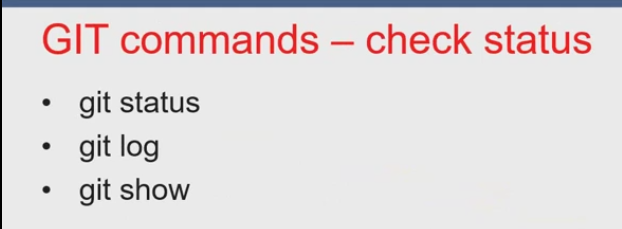
****

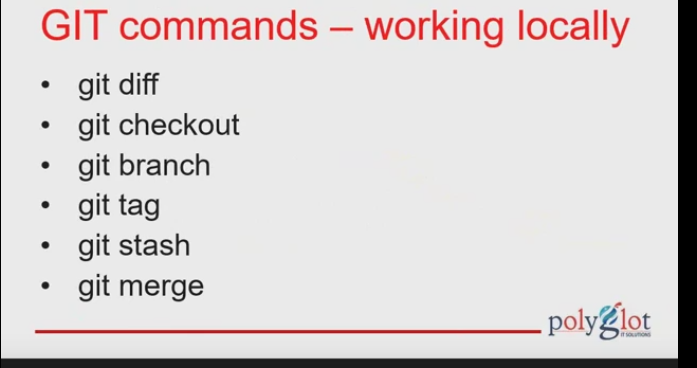
**# git config --list**

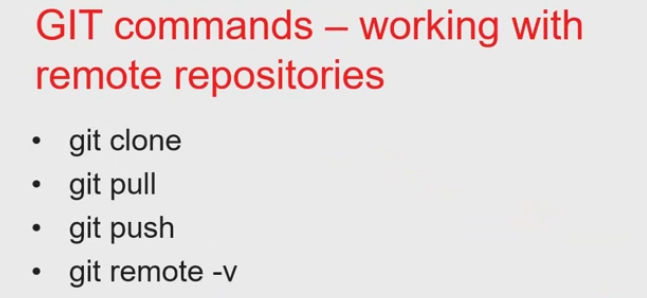
****

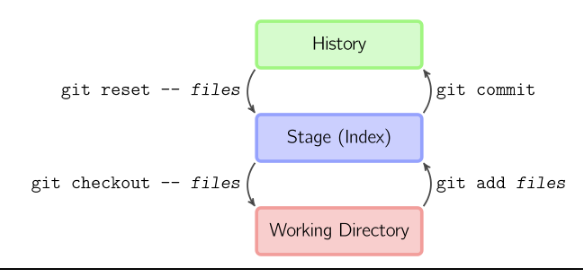
**# git config --global color.ui true**

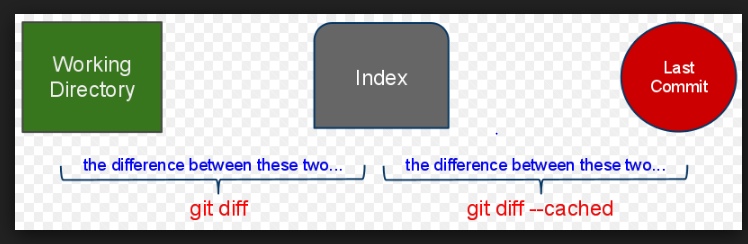
****

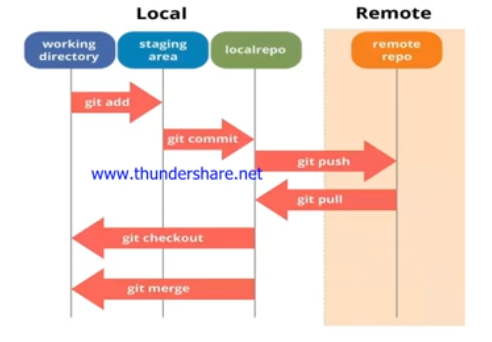
****

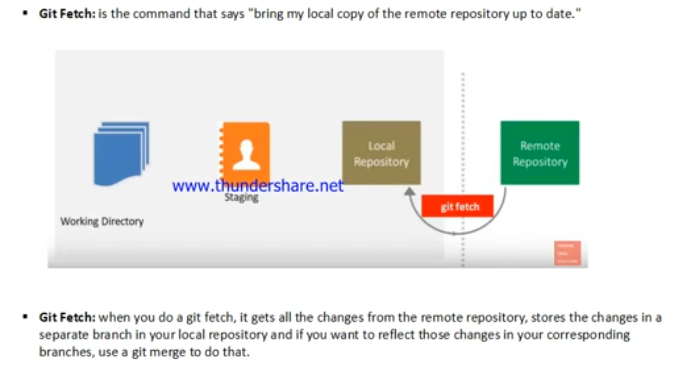
****

****

****

****

****

****

**Git merge have two types of merges**

1. **Fast-forward merge** : move branch directly to master branch
2. **3-way merge** : not a direct path to merge master branch to merge branch

**Interview quest) what is the diff between git-clone and git-init?**

**Interview quest) what is the branches and how to create branches?**

**gitk :** gitk is a graphical history viewer

**git gui :**

**GitHub for Mac and Windows**

**Git in Visual Studio**

**Git in Eclipse**

**Git in Bash**

**Git in Zsh**

**Git in Powershell**

**Sai siva**

**.gitignore:**  on any workspace. in which you want to ignore some files. Then create file in the name of .gitignore and inside this .gitignore file we put names of files. Whatever files you want to ignore then Git will be not consider those files

**Git cheery-pick:**  you will have to specify what particular commit-ID that you have to merge to destination branch

Pick to specific commit-ID that one is merge to branch called cheery-pick

**Git merge:** whatever file have in source branch that all files in a single step completely merged to destination branch

****

**In git, sometimes we don’t want to commit our code but we do not want to lose the unfinished code. In this case we use the git stash command to record the current state of the working directory and index in a stash.**

**This stores the unfinished work in a stash, and clean the current branch from uncommitted changes. Now we can work on clean working directory**

**Later we can use stash and apply those changes back to your working directory. At times we are in middle of some work we do not want to lose the unfinished work, we use git stash command**

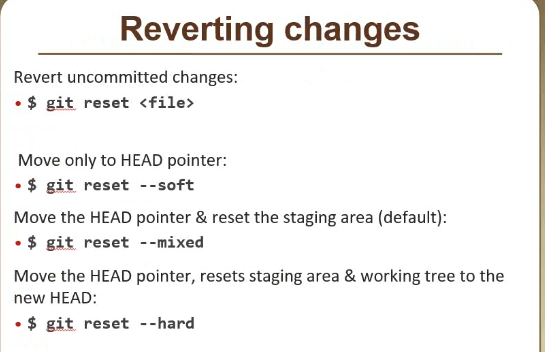
**Git stash**

**When you have done a change in a file or multiple files & before going to do staging area or committing it & if you feel that you want to take back up then you will do a git stash**

**Where git stash does 2 things that is**

1. **it will take backup of all latest modification files and then**
2. **it will undo the changes the from the file**

**REVERTING**

****

**Before going to the commit you are actually trying to do a undo/revert the changes**

**If you want to undo, whatever changes you done before committing, you will be give a command called git revert**

**Peer review 🡪 what ever you wrote code that code is reviewed by your team mates**

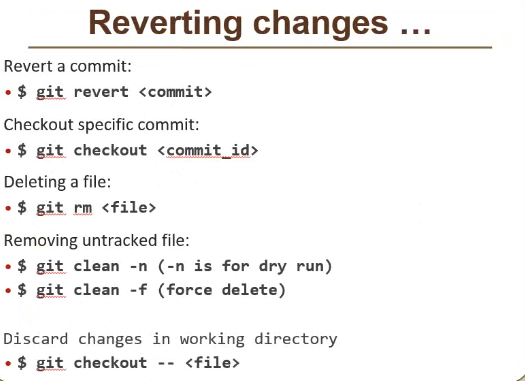
**When files or data moves to staging area using ( git add . ) then one temporary commit id is created in local repository and it will be tracking it**

**Git reverting is undo the changes 3 areas (working directory + staging area + temporary ID )**

**# git reset --soft ( only temporary ID is deleted. HEAD will back to previous commit ID)**

**# git reset --mixed ( here undo the changes temporary ID as well as staging area )**

**# git reset --hard ( here also undo changes from all 3 areas i.e., temporary ID + staging area + working directory )**

****

**# git log --oneline**

**# git checkout commit-ID 🡪 To Move from one commit ID to other commit ID**

**# git checkout branch-name 🡪 switch one branch to another branch**

**# git checkout branch@@commit-ID**

**Particular commit id is associated with branches. Under repository have branches**

**Commit ID 🡪 branches 🡪 Repository**

**Commit id branches repository**

**HEAD is always point out latest commit-ID**

**# git rm filename 🡪 it means remove the file from both current working directory and also local repository and then add it in to the staging area.**

**Untracked** - This **file** exists locally, but isn't a part of the **Git** repository.

For example some changes in **tracked file** that are not yet in staging area. You can start **tracking**using **git** add, a **file** directly goes to staging area when you use **git** add, now that **file** is both **tracked**and staged. You can unstage any staged **file** using **git** reset -- filename. An **untracked file** is in unstaged state also.

**Untracked files 🡪 means files are not available in local repository**

**# git clean 🡪 from workspace it will delete the all un tracked files**

**# git clean -n 🡪 it show what going to be deleted simply called as dry run**

**# git clean -f 🡪 forcefully delete the all un tracked files**

****

**Tag is nothing but a string**

**A name we will attached to the commit ID.tag is nothing but a name convention. tag will be used for identification**

**For example**

**# git tag -a Release3.1.2 -m “test commit for release….3.1.2” commit-ID**

**# git show <commit ID>**

**Workspace = working directory + staging area + local repository**

**.git = is the git repository of your workspace**

**Central repository 🡪 store + share**

**Local repository 🡪 store + share + modify**

**Triggers/hooks/wrappers script in git:**

**Cd .git --- cd hooks ---- ls**

**Pre-step / pre-script / pre-hook**

**Post-step / post-script / post-hook**

**Github 🡪 online place is storing your repository**

**Author sai siva—**

**Software Configuration Management**

SCM is a process of tracking and controlling changes in the software

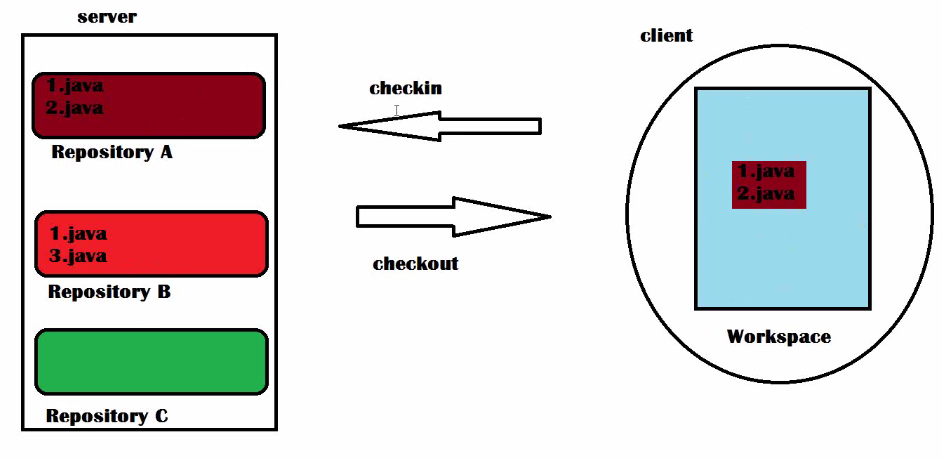
the goals are generally

team work

defeat tracking

configuration auditing

process management



**what is a repository?**

place in a server where all files related to a one product are stored and maintained.

**what is a container?**

place where store something. It is also called repository.

**what is a workspace?**

it is a directory where i will working on the files of a particular repository.

the interaction between the server and the directory is having by using the client(software).

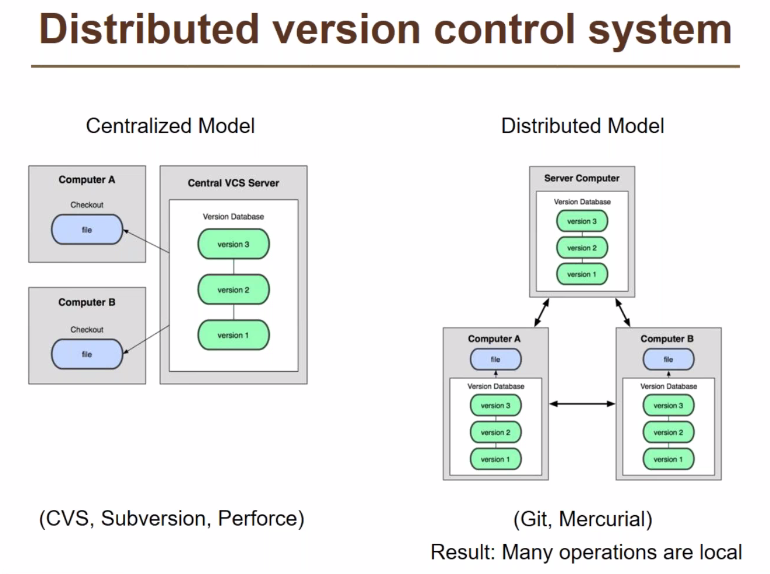
**what is checkout?**

it is a process where we will see the files inside the workspace on your machine corresponding to the project

**what is checkin?**

after modifying the files in workspace and updating the files into the server.

**VERSION CONTROL SYSTEM**

****

**Centralized VCS:-**

Disadvantages:-

Always connected to the server.

If server and client are in different zones the time taken for checkin and checkout is very high.

Time taken for checkout is very high.

If computer A has to make any changes then it has to checkout first and updates the file and checkin into the server and compuer B has to checkout to see the changes made by the computer A.

**GIT**

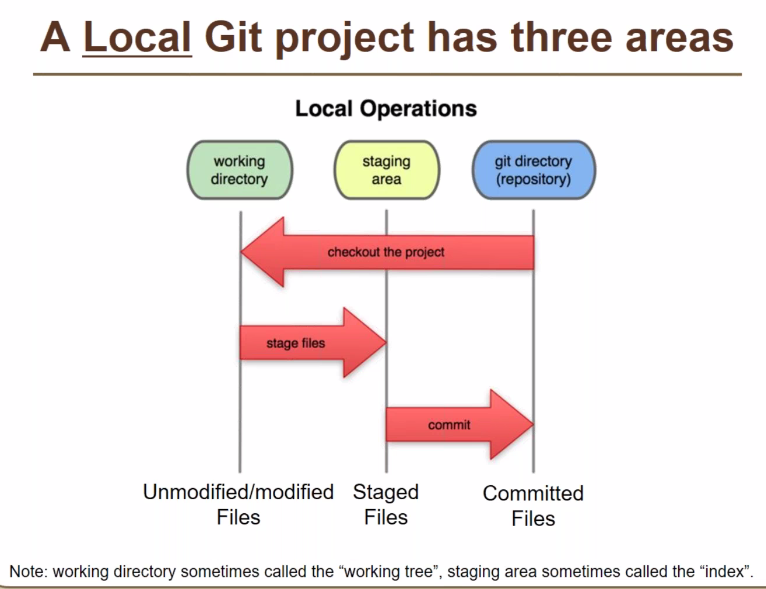
In Git everything is a workspace.

**What is snapshot?**

At some point of time what do you exactly have.

**What is commit id?**

Commit id is nothing but a 41 characters uniq id which is created by the git and stored by git, having details about who is modified, what is modified and when it is modified. Snapshot of that particular change.



In every workspace it consists of three areas

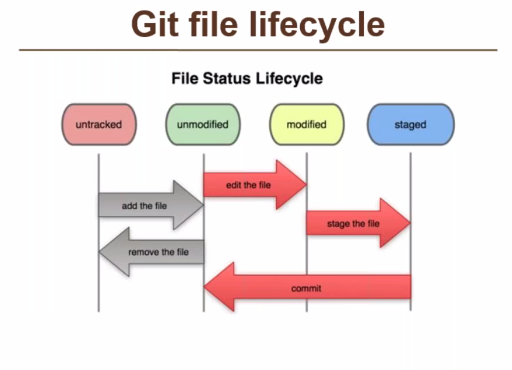
Working directory 🡪 where we make any changes like modifing files. We can see the files and modify the files in workspace is only in working directory.

Staging area 🡪 it is a buffer there is no directory in workspace but logically present

Git repository

In Git checkin is two steps one is adding into stageing area and then commiting.

In working directory we modify the all the files and add into the stageing area and from staging we commit it create the snapshot of the what you already had and what you modified.



Untracked means when we are creating a new workspace i.e., first time in the project the repository in the workspace is empty. When we add files into the working directory is called untracked files because there is no information about the files in the git repository.

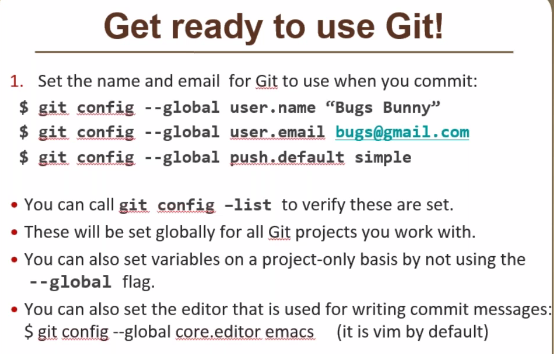
After modifing we added into the staging and commited into the git repository then the content in the working directory and the repository is same then we called as unmodified.

Modified means when we modify the files in the working directory and there is a difference in the content of working directory and repository.

git config --global user.name “sai”

--global 🡪 we can use more than one 1 workspace and all the workspace uses the same username

If you want to use the different username then you have to set locally instead of global.



mkdir central.git

cd central.git

git init --bare 🡪 server where we only store and manage

cd ../

git clone central.git workspace1 🡪 creating workspace and copying the content from the server.

cd workspace1/ 🡪 workspace where client modify and edit the files.

touch 1.java

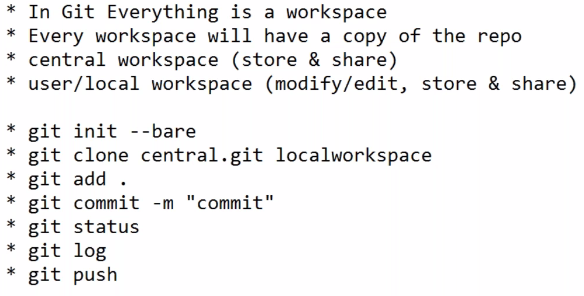
git status

git log

git push 🡪 uploading the commited files into the server.

cd ../central.git

git log 🡪 we check the files uploaded into the server.



When we are pushing any commits to the server ensure that all the commits in the workspace is there in the local repository else it shows error.

git pull 🡪 only the new commits which are not in the workspace repository is downloaded.

git log 🡪 shows all the commits from latest to the old.

git log -3 🡪 latest three commits will display

git log --oneline 🡪 displays only the first 7 characters of all commits and commit messages.

git log --oneline -2 🡪 displays the l

atest 2 commits only in short form.

git log --grep “file” 🡪 displays all commits with file commit message.



If you want to ignore some files while adding into the repository then you have to add the file names into the .gitignore file.

Vim .gitignore

Add the file names, those files are excluded.

What is parallel release?

It is the same product and same functionality releasing once seperately for android,windows and mac.

**Branching:-**

It will help us to do parallel devlopment.

Branching means for the same product if you want to do different release with small changes in the content of the product ,you are going to do the release and everything is one product and one repository.

When we create a repository by default we are not storing commit id directly into the repository, but you are creating a branch/layer only on which commit ids are storing.

In git default branch is master.

Branch is a place in a repository where commit ids are stored.

git branch 🡪 on which we are working.

When we are switching from master branch to another branch whatever the content in the working directory will be cleaned and the new branch content is stored into the working directory, after any changes we added into the staging area and to repository of that particular branch is stored.

ADVANTAGES:-

No need another repository for parallel branch.

Any no of branches can be created.

These data is stored in the local repository unless you shared into the server.

You can test on your own no need to share others.

No need to ask administrator for the repository we can create a branch on its own.

It is not possible in other version control systems except in git.

git branch <branchname> 🡪 new branch will create

git branch 🡪 shows all the branches in the repository.

\* -🡪 indicates the present working branch.

git checkout <branch name> 🡪 to switch to another branch.

If branch is created from the master the content in the branch is the replica of the master, going forward only the changes is stored in the branch. Master branch changes is stored only in the master.

By default when we are push into the server, it is from master in the local repository to the master in the server.

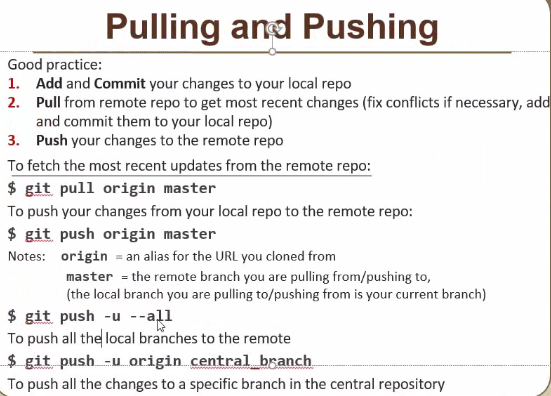
When you are trying to push from the new branch which is not in the git it won’t push.

**How we will share the data from new branch to server?**

Just by merging the two branches.

git merge <source branch> <destination branch>

After merging it will create one more commit id regarding the merging.

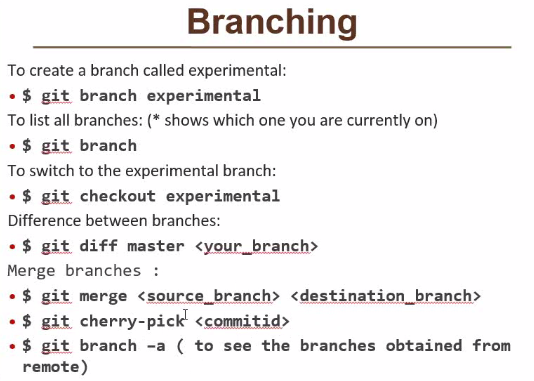


git push -u --all 🡪push master and new branch into the repository.

When ever now trying to creating a workspace it will defaulty shows the master branch in the local repository. By checking the git branch. But it shows the reference, if you want the complete branch then,

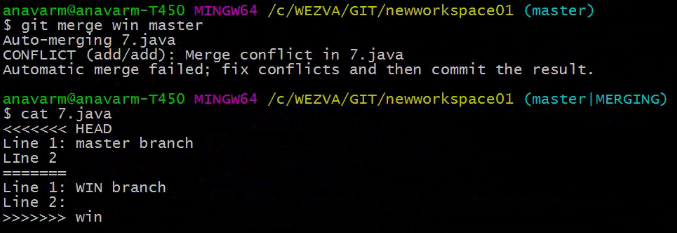
git checkout <branch name> 🡪 it will remove the reference and create the branch in the local workspace.

git branch 🡪 it will show both branches.



git cherry-pick <commitid> 🡪 when we are merging all the commit ids which are not in destination will be taken in a single shot. In order to add only one commit id cherry-pick is used.

When we are merging the branches, if we have the same files with different data in that files then there is conflict we have to manually solve these problems. It will show in that manner,



Content above the ‘===’ is destination, below is source.

We has to manually edit the file whatever the content we want. There are some tools used on the top git to resolve the conflict issues but it uses same logic but everyone won’t need the same logic.

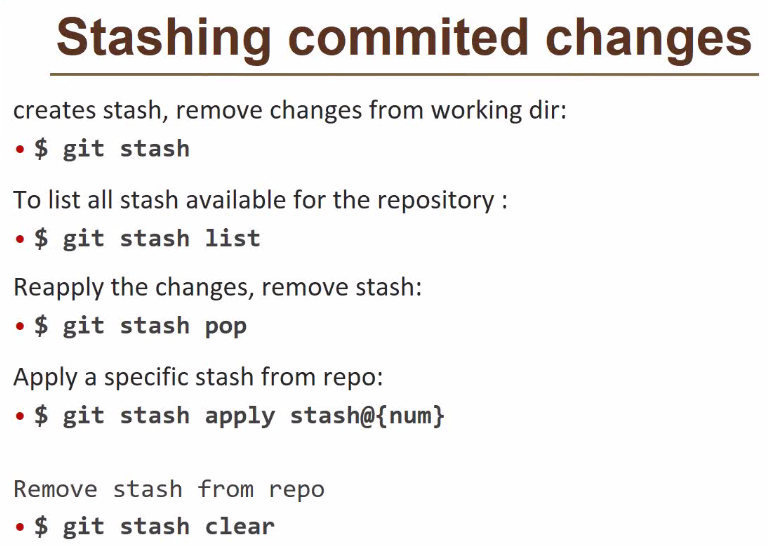
**STASHING IN GIT**

When we writing scripits we implement one idea, but it is not upto the mark. Then we try to use another idea, but we won’t guarntee that second idea will be successful or not. So we have to keep backup the first idea changes.

Advantages:-

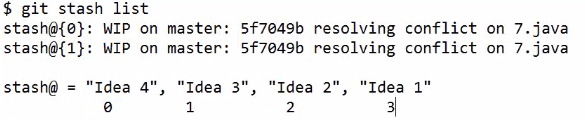
Try to take backup of all files that we made changes before the commit.

After backup it will undo the all changes in the file and start writing from the beginning.



What is an array?

Array is an variable where we store multiple values and we can retrive.

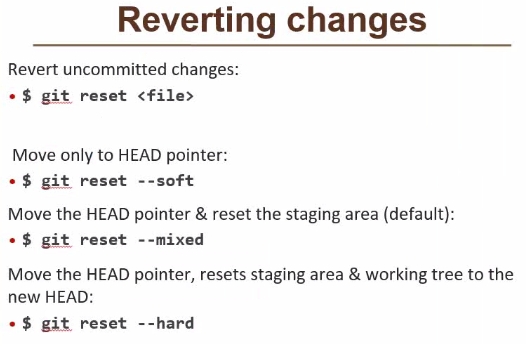


0,1,2,3., are index numbers.

0 for the latesh backup/idea, if you want to restore a particular idea we have to use that particular index number.

What is head of the branch?

We are having the different commit ids, on which commit id we are working and making changes is called the head of the branch.



Reverting means the undo the changes without doing backup.

As per user prospective reverting means undo the content in the file what was modified above the content of the file, but actually we can’t delete any content in the file we are doing one more commit with the previous commit data.

What is tracking?

At any point of time I go back and check what we had at a particular point of time.

Checkout and reverting both are different. Checkout means we are seeing the content in the working directory at a particular commit id, it means changing the head.

**git checkout current@@<commit\_id>**

**git checkout <branch>@@<commit\_id>** 🡪 if we won’t mention any commit\_id default it will goes into the latest commit\_id.

Actually we have mention like this, if we mention without branch it will default looks in the current working branch.

When we checkout to a particular commit-id, then we check the **git log** it will show the commit-ids below this only. This doesn’t show the commit-ids above this, if we want to see the all commit-ids then we have to use git checkout <branch name>. then it will display the all commit-ids.

**git rm <file>** 🡪 it will remove the file and automatically added into the staging area we have commit only, it will create new commit id.

**git clean 🡪** to clean the untracked files i.e., what ever the files that are created first time if won’t need then we delete that files, when developer checks the files it creates some logs, these logs for us is not needed to push into repository.

**git clean –n** 🡪 it will show the files which will remove in the workspace. If any files are needed then we have to commit these files.

**git clean –f** 🡪 delete the untracked files forcefully.

**TAG**

What is the purpose of tag?

Generally we use tags for the releases, to notify or identify the release and what commit id we were giving to the QA for the testing.

**git tag –a <pattern> -m ‘comment’ <commit\_id>** 🡪 to create commit id.

**git show** 🡪 to show the contents of the tag.

**git tag** 🡪 to show all the tags.

**git tag –d <tag>** 🡪 to delete the tag.

**git push –tags** 🡪 to push the tags into the central repository instead of commit\_ids.

One commit\_id has one or more tags, whereas one tag cannot applied to multiple commit\_ids.

**Working directory** is a place where we are see the files physically and modify.

**Staging** area is a buffer place which will not visible.

Repository is a directory where we are storing everything. 🡪 **.git**

**GITHUB**

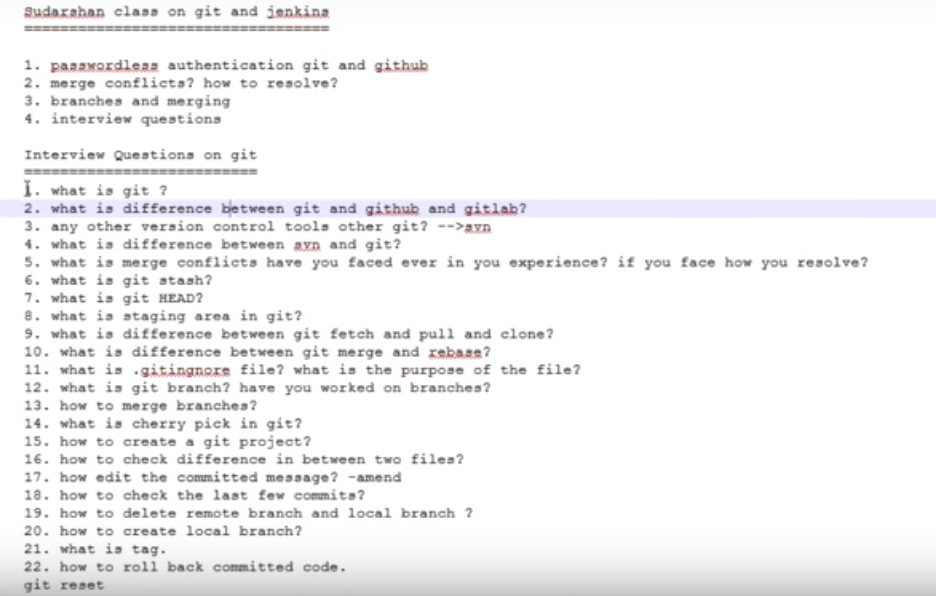
Github.com is a site for online storage of git repositories.

You can get free space for open source projects or you can pay for the private projects.

**Triggers/hooks:-**

Simple script we can use in which you can try to put some process and we can apply that script to a particular command.

**GIT INTERVIEW QUESTIONS:**

****