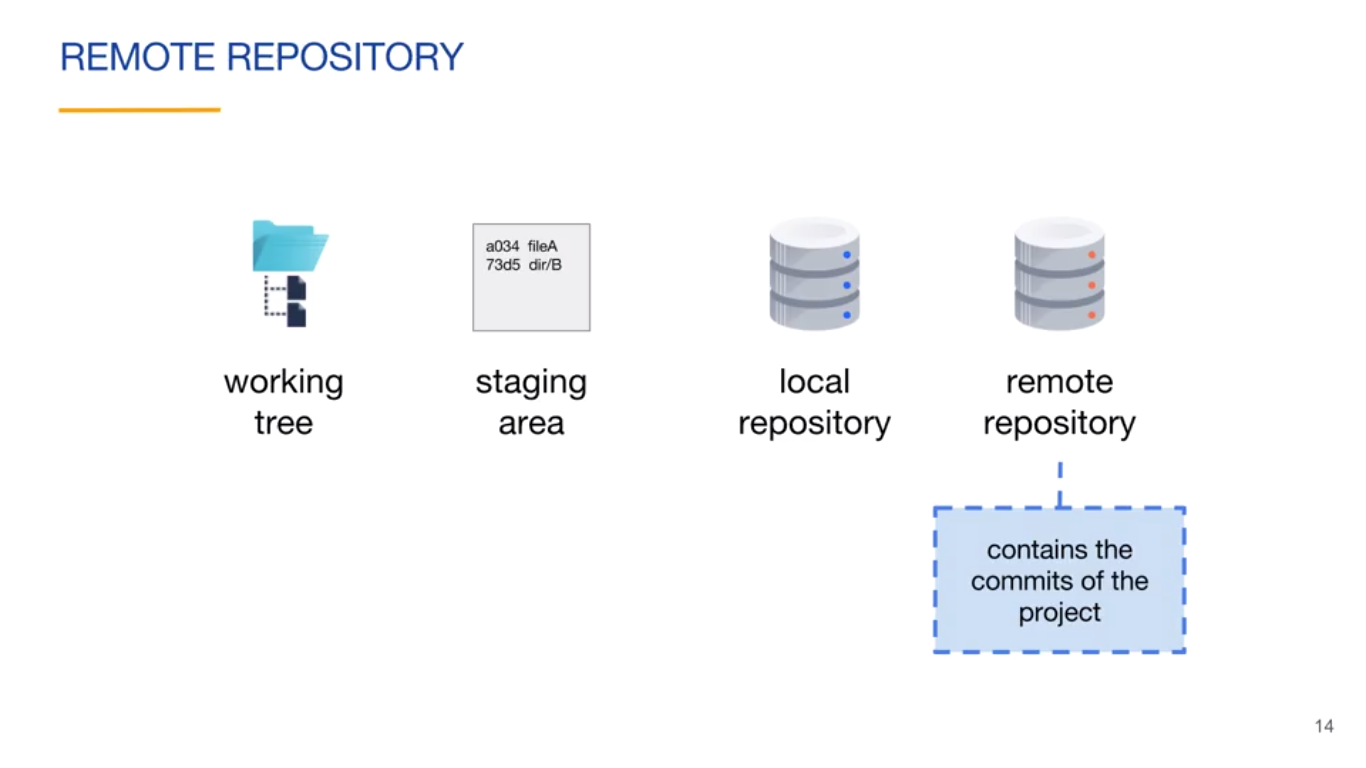
GIT Version Control System

Commits: This is a simple snapshot of the whole project but will not contain all the files of the project again and again but will just contain the improvements in comparison to the previous one.

Branching: This is the ability with the vcs in order to work independently and we can create a branch from the master branch and then we can work from there and then after making per say a new feature, we can introduce it back into the master branch using the pull request and then branching will be over.

Pull Requests: This means that we have pulled the branch into our final

Version Control System: These are types of systems which are used for maintaining the history of the work. And git is one of the developed system for the same.

And it can be used 

Working Tree: It is basically the folder where our files are located.

Staging area: this is the folder which is going to be the folder which contains the files ready for next commit.

Local repository: This contains all the commits of the project.

Remote repository: It contains the files which are currently official.

**Create a local repository :** git init

It will initialize the directory with .git directory which contains the staging area and local repository which is used for storing the history of the commits.

After creating the local repository our local repo will be having all the necessary files required for the git to handle the version control system of the project.

The commands which are used for working of the git are:

1. **Git status:** which describes the files present in the working directory for the commit.
2. **Git add file\_name/ .** : this will make the untracked files into the staged files.
3. **Git commit –**m ‘COMMIT\_MESSAGE”: Adds the message to the last commit for reference.
4. **Git log:** this describes the history of all the commits done for the current repository.
5. **Git log –**oneline: this will give the condensed form of the log and if we want certain number of recent logs then we can do the same by appending that number at the end.

**Pushing to the remote repository:** We can push our history of changes to the remote repository in two ways, if we have transactions done in our local repo then we can push them to remote repo by adding the remote repo to our local repo and then pushing it to the remote repo. And other way to complete the same task is by cloning the remote repo into our local repository.

**Cloning the remote repository:** In this we can clone the remote repo onto our computer using the link provided by the repository host such as **github.com**.

When we clone a remote repo from a website then repository will be added automatically to the local repository and whenever we push then it will be pushed to the added repository.

**git remote –v:** it displays the content to which our local repo is linked to remote repo and there will be an alias related to the local repo and by default it is named as **origin**.

**$ git remote -v**

**origin https://github.com/vishal22091997/Basic-Programming.git (fetch)**

**origin https://github.com/vishal22091997/Basic-Programming.git (push)**

**Adding the remote repository to the current local repository:** In this we need to add the remote repository using git add command

**Git remote add GIT\_URL\_NAME GIT\_URL:**

**For example:**

**git remote add origin\_1** [**https://github.com/vishal22091997/testing\_.git**](https://github.com/vishal22091997/testing_.git)

**git remote add origin\_2** [**https://github.com/vishal22091997/remote\_repo2.git**](https://github.com/vishal22091997/remote_repo2.git)

in this case we have linked github remote repo to our local and we have linked our local repo to two different remote locations so while pushing we will be having option to push it to the repos where we want it to.

**For pushing it to the repo:**

**Git push –u origin\_1 master**

This simply means that the branch which we want to push on the commits of the master branch to the origin\_1 i.e. testing\_ repo.

**GIT ID’s**

These are the SHA-1 values which are to be there for the commit and these values change very much on changing the content even very slightly.

**References:** In this we need to use either the commit id i.e. part of it or full of it or using the references in order to refer and see that particular commit.

GIT Id’s are the SHA-1 values of the content on which they are being applied and as the content changes the value of GIT ID will also change.

SHA-1 value is meant to be Avalanche which means that very small change in the content leads to the very high change in the values of ID’s

**Git hash-object FILE\_NAME**

This will generate the SHA-1 value generated due to the corresponding file.

This hash-object is a pluming function which is used by the git but not by us

**Git log –oneline**

This will return the shorten version of git id.

**Git show SMALLEST\_FRONT\_PART**

In this we can add the front part of the ID and let it judge the part to which it refers.

**REFERENCES**

When we simply create the master branch and start working on it then or master branch is pointing to the tip of the commit. And we can also check where our head branch is actually pointing to by entering into .git/refs/head/master this file master content is nothing but the git id of the commit to which master head is pointing.

Our head reference keeps on pointing to the master and master contains the git id of the last commit which simply means as follows.

If we go and see the HEAD file which is present in the .git/HEAD directory we can see that it contains the address of the master pointer.

~ and ^ are the two symbols which we can use in order to refer to the parent of the current pointer and ^ for referring to the parent of the merged commit.

Now Coming to the tags which we can associate with the commits and then we can use these tags to show the log

Like git show v0.1

Git tags: it shows all the tags which are present in the project.

Git tag TAG\_NAME POINTER\_TO\_WHICH\_WE\_WANT\_TO\_TAG

Example: **git tag v1.0**

As we haven’t given any pointer so this will be the tag for the current commit to which head points.

**Git tag v0.1 HEAD^**

This will tag the head’s parent as v0.1

1. LIGHTWEIGHT tags

Git tag <tag\_name> [<commit>]

1. Annotated tags

Git tag –a [-m <tag\_msg> | -F <msg\_file>] <tag\_name> [<commit>]

For example: git tag –a –m “includes feature 2” v0.1

So this includes the tag msg with current commit to which head points and the tag name is v0.1

This is just for adding the tags to the local VCS but if we do push to the content then this will not add tags into our push but we specifically need to go and do this task by us.

Git push <remote\_name> <tag\_name>

Git push <remote\_name> --tags #for transferring all the tags in one go.

Master