**Version Controlling:**

* This is the process of preserving all the versions of code in remote server, All the team members upload their code (checkin) remote version controlling system, The vcs accepts the code uploads from multiple developers and create in integrated projects from these uploads, The next time developers download the code from the version controlling system, it will contain all the code created by the entire team,
* Vcs ‘s also maintain all different versions of the code so that team members can switch btw any version according to the requirement
* Vcs also keep a track of who is making what kind of changes
* There are 2 types of vcs 1.centralised 2.Distributed
* **Centralized version controlling:**

Here we have a remote server into which all the team members upload the code, version controlling happened only on the server.

On the individual developers machines only code is present.

**Code developer 1**

**de**

**Code**

**developer 2**

* **Distributed version controlling:**

Here we have local repository installed in every developers machine, initially code created by developers is uploaded into local repository where version controlling happens at the level of individual developers, from here the code is pushed into the remote repository where version controlling happens at the level of entire team

**Remote Server for entire Application**

**Remote Server for PB**

**Remote Server for Laavi**

* Another advantage of distributed version controlling is we can maintain sub repositories at the level of individual smaller team, These repositories are called “ Bare repositories”
* Installation of Git on windows:

1. Open <https://git-scm.com/downloads>

2. Download git for windows-Install it

3. Once git is installed we get an application called git bash which is the command prompt of git

* Installation of Git on Linux:

1. Open terminal in a linux machine
2. Update the apt repository

Sudo apt-get update

1. Install git

Sudo apt-get install –y git

* Configuration of git into machine:

1. Setting username and email globally for all users

git config –global user.name “sai krishna”

git config –global user.email [selenium.sailsrishna@gmail.com](mailto:selenium.sailsrishna@gmail.com)

* Git when working on the local machine uses three components

1. Working Directory or workspace
2. Stagging Area
3. Local Repository

* Working directory is the default location where the developer creates the code, Initially all the files present in the working directory are called untracked files
* Staging Area is an intermediate buffer zone into which files are initially moved before version controlling start. The files present here are called staged files
* Local Repository is the locations where version controlling happens at the level of the individual developer. The files present here are called committed files.

1. To initialize the working directory into a git repository

cd path of\_working\_directory

git init

The above command create a hidden folder called “.git” where it stores the configuration that the necessary for git to run.

1. To send the files from working directory to staging area

git add filename

1. To send multiple files into staging area

git add file1 file2 file3

1. To send all files and folders including subfolders into staging area

git add .(dot) – dot represents current working directory

1. To bring files back from staging are to untracked sections

git rm –cached filename

git reset filename

1. To send files from staging area to local repository

git commit –m “some message”

1. To check the status of the untracked and staging areas

git status

1. To see the commit history of the local repository

git log

1. To see the commit history in oneline format(only few matter)

git log --oneline

* .gitignore:

This is a special hidden configuration file which is used for storing private files info. The file names that we store .gitignore will no longer be accessed by git

1. Create few files in the working directory

Touch file1 file2 file3 file4 file5

1. Check the git status

git status

All the above 5 files will be shown as untracked files.

1. Imagine the first four files are private and they should not be accessed by git

Cat > .gitignore

file1 file2 file3 file4

To come out of cat command ctrl+d

1. Check the status of git now

git status

It will show only file5 and .gitgnore as untracked files file1-file4 are not longer accessed by git

* **Branching in git:**

This is the feature in git which is used by devops for creating code in different functionality of code on different branches by default, git performs all activities on master branch ,advantages of using branches is code can be created in unclutted way and later code can merged with master branch

1. To see the list of local branches

git branch

To see all the branches (local and remote)

git branch –a

1. To create a new branch

git branch branch\_name

1. To move into a branch

git checkout branch\_name

1. To create a new branch and also move into it

git checkout –b branch\_name

1. To merge a branch with master first move to master and then merge

git checkout master

git merge branch\_name

1. To delete a branch that is merged

Git branch –d branch\_name

This is also called as soft delete

1. To delete a branch that is not merged

git branch \_D branch\_name

This is called as hard delete

**Note:** Whenever a commit is created whatever is the commit history of master till that point will be copied to the new branch

**Note**: Irrespective of where a file is created or modified git always considers only the branch from where it is commited and that file belongs only to the branch where the commit happened.

* **Git merging:**

Whenever branch merged with master branch the commit history repsent on the branch we get merged with master branch based on time stamp

1. Create few commits on master

touch f1 f2

git add .

git commit –m “a”

1. Create a new branch called test and create commits on it

git branch test

git checkout test

touch f3 f4

git add .

git commit –m “b”

1. Check the commit history of the test branch
2. Move to master branch and create fe more commits

Git checkout master

Touch f5 f6

git add .

git commit –m “c”

1. Check out commit history of master branch
2. Merge test branch with master branch & delete test branch

git merge test

git branch –d master

1. Check commit history of master branch

* **Working on remote git repository(github):**

1. Open github.com--🡪sign up for a free account
2. Sign in into that account
3. Click on Plus on top right corner -🡪Click on new repository
4. Enter some name for repository---> select public

Click on create repository

1. Go to “Push and existing repository from command line”

Copy the 1st command

Paste it in git bash

This will create a link between the local repository and the remote repository

1. Copy the second command and paste

git push –u origin master

Enter username and password of github account

The entire code from the local repository will be uploaded into the remote github repository. This is called checkin.

1. Upload the new commits(files) in existing repository in github

git push –f origin master

* **Downloading code from remote git repository:**

This can be done in 3 ways.

1. git clone

2. git fetch

3. git pull

* **git clone:**

This will download the entire remote repository into the local machine irrespective of weather that code is already present on our local machine or not.git clone is generally one time activity where the team lead creates the basic folder structure and uploads into the remote github. All the team members run the git clone command to download the entire remote repository into their local machine.

Synax:

Git clone remote\_repository\_url

* **git fetch:**

This will work only when there are modifications in the code i.e the code present on the remote server is different from the code present on the local repository. Git fetch downloads only the modified data and it will place that data on a remote branch. We can checkin into that remote branch and see the modifications. If the modifications are acceptable we can merge then with the master branch

1. Open github.com
2. Click on the remote repository that we uploaded
3. Select some file to be modified-🡪click on it
4. Click on Edit icon-🡪 make some changes to the file
5. Click on commit changes
6. In git bash

git fetch

The above command will download the modified files and place it on a remote branch.

1. To see the list of all the branches

git branch –a

1. Move to the remote branch

git checkout remotes/origin/master

1. See the modified files and if they are acceptable merge with master

git checkout master

git merge remotes/origin/master

* **git pull:**

This will also work only when there are modified files on the remote server but it will merge those files directly with the master branch

1. Open github.com
2. Click on the remote repository that we uploaded
3. Select some files to be modified -🡪 click on it
4. Click on edit icon -🡪 make some changes to the file
5. Click on commit changes
6. In git bash

git pull

we can see the modified files directly on the master branch

* **git rebase:**

This is feature of git which is used for performing fast forward merge i. e commits coming from branch will be projected on the top most section of master branch head will point to the top most commit which comes from branch

1. Create few commits on master

touch f1

git add .

git commit –m “a”

touch f2

git add .

git commit -m “b”

1. Create a new branch called “test ”and create few commits on it

git checkout –b test

touch f3

git add .

git commit –m “c”

touch f4

git add .

git commit –m “d”

1. Move to master and create few more commits

git checkout master

touch f5

git add .

git commit -m “e”

touch f6

git add .

git commit –m “f”

1. Rebase test branch with master branch

git checkout test

git rebase master

git checkout master

git merge test

1. Check the commit history of master

git log --online

* **Git cherry pick:**

This is used for choosing which commits we want to take into the master branch generally when we perform “git merge ” or “git rebase ” all the commits of that branch will come into master branch

Cherry pick will allow us to select only those commits that we require and merge then with master

1. Create few commits on master

touch f1

git add .

git commit –m “a”

touch f2

git add .

git commit -m “b”

1. Create a new branch called “test ”and create few commits on it

git checkout –b test

touch f3

git add .

git commit –m “c”

touch f4

git add .

git commit –m “d”

touch f5

git add .

git commit –m “e”

1. Check the commit history of test

git log --oneline

1. Identify the commits that we want to copy to master
2. TO cherry pick those commits

Git checkout master

Git cherry-pick commit\_id1 commit\_id2

* **Git stash:**

This is a feature of git which is used for leaving unfinished work and start a new functionality related coding. Further commands of git should not touch the unfinished files

1. To stash the staged files

git stash

1. To stash the untracked files and staged files

git stash –u

1. To stash the .gitignore ,untracked files and staged files

git stash –a

1. To see the list of the stashes done

git stash list

1. To bring the latest stash out from the stash area

git stash pop

1. To bring an older stash out from the stash area

git stash pop stash@{stash\_number}

* **Git amend:**

Whenever we modify a file or create new files generally we create a new commit. Instead we can put the modifications in the existing commit itself rather than creating a new commit

This can be done using git amend command

1. Check the commit history

git log –oneline

1. Modify some file or create new files

touch f10 f11

1. Send them to staging area

git add .

1. Instead of creating a new commit we can add this to the top most commit

git commit –amend -m “To most commit message”

1. Check the commit history

git log – oneline

**Note:** git reflog

* **Tagging in git:**

Tags are used for placing bookmarks on commits. They are to specify info related to who tagged, when it was tagged and why it was tagged. Generally used for releases. This helps in understandings what are the commits that are related to specific releases

Tags are 2 types

1. Light weight tags

2. Annoted tags

1. To see the list of all tags

git tag

1. To create a lightweight tag

git tag tagname

1. To create an annoted tag

git tag –a tagname –m “some message”

**Note:** Tags are always created or the topmost commit

1. To create tags for an older commit

git tag –a tagname –m “some message” older\_commit\_id

1. To see tag info

git show tagname

1. To delete a local tag

git tag –d tagname

1. To push all tags in git remote repository

git push –tags

1. To delete tags from the remote git repository

git push origin : tagname

* **Git reset:**

Reset command can be used for moving files from staging area to working directory and also from local repository to staging area or working directory it prefer 3 ways soft ,mixed and hard

**Soft reset**: files present in topmost commit will be sent to staging area, git status will show these files which moved topmost commit early as coming back to staging area but working directory will not affected. i.e it will show files as they were present at time of latest commit.(one step back)

Syntax: git reset - -soft commit

**Mixed reset**: it will directed sent the files local repository to working directory they will shown either modified or untracked files but still files contain code as present in latest commit

(Two steps back)

Syntax: git reset - -mixed commit

**Hard reset:** it will move head to older commit and it will also make changes in file as they were present in the time of older commit (three steps back)

Syntax: git reset - -hard commit

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

**Jenkins:**

This is a tool for performing CI-CD

**Stages in CI-CD**

**Stage 1 (Continues download):**

Developers creates code and upload that code into the version controlling system like Git, Jenkins is integrated with the git version controlling system in such a way that whenever develops makes any modifications to the code and uploads into Git, Jenkins gets a notifications and it downloads that the code, This is called Continues Download

**Stage 2 (Continues Build):**

The code downloaded in the previous stage has to be built and we have to create an artifact. This artifact can be in the format of war.ear exe file etc, Jenkins performs this build process using plug-in like Maven plug-in, Ant plug-in, MsBuild plug-in etc, This stage is called continues build

**Stage 3 (Continues Deployment):**

The artifact created in previous stage has to be deployed in the QA Environment. The Testing servers might be running on some applications servers like tomcat, jboss etc. Jenkins will now deploy the artifact into these applications servers. This is called Continuous Deployment

**Stage 4 (Continues Testing):**

Once the applications is deployed into the testing environment Jenkins run the automation testing programs (selenium) created by the testers and check if the applications is working correctly. If the selenium programs show any errors the developers again fix the defects and upload the modified code into the Git. Jenkins will start all the above 4 stages again.

**Stage 5 (Continues Delivery):**

If the automations testing programs pass Jenkins will now deploy the applications into the production environment where the enduser or client can start accessing it .This deployment into production environment is done by Jenkins after taking approvals from the delivery team. This stage is called continues delivery.

Note: The first 4 stages are called continuous integration the 5th stage is called continuous delivery

**Set and Installation of Jenkins:**

1. Update the apt repository

sudo apt-get update

1. Install java 8

sudo apt-get install –y openjdk-8-jdk

1. Install git and maven

sudo apt-get install –y git maven

1. Download Jenkins.war

wget http://mirrors.jenkins .io/war-stable/latest/Jenkins.war

Note: (<https://jenkins.io/dowmload/> -----🡪Long term support ----🡪Generic java package)

1. To start Jenkins

Java –jar Jenkins.war

1. To access Jenkins

Launch any browser

Public ip of Jenkins\_server:8080

1. In the unlock Jenkins screen enter the initial admin password -🡪 continue
2. Click on install suggested plugins
3. Create first admin user -🡪continue
4. Click on start using jenkins

**Setup of tomcat on QA Server and Production Server:**

1. Connect to QA server instance using gitbash
2. Update the apt repository

sudo apt-get update

1. Install tomcat8

sudo apt-get install –y tomcat8

1. Install tomcat8-admin

sudo apt-get install –y tomcat8-admin

1. Open tomcat-users .xml file and set the credentials

sudo vim /etc/tomcat8/tomcat-users.xml

go into insert mode by pressing i

Enter the below statement

<user username=”intelliq” password=”myintelliq” roles=”manager-scripts”/>

Save and quit Esc : wq enter

1. Restart tomcat8

sudo service tomcat8 restart

**Stage -1 (Continues Download):**

1. Open the dashboard of Jenkins
2. Click on New item --🡪 enter item name as “Development”
3. Select Free style project --🡪 ok
4. Go to Source Code Management—select git
5. Enter github url where developer has upload the code

https://github.com/sabithakota/practice

1. Click on Apply --🡪save
2. Go to the dashboard of Jenkins---🡪 go to Development job ---🡪click on build icon

The above job will download all the code uploaded by the developer into the github repository

**Stage -2 (Continues Build):**

1. Open the dashboard of Jenkins
2. Go to the Development job----🡪Click on Configure
3. Go to Build section
4. Click on Add build step
5. Click on Invoke top level maven targets
6. Enter the maven goal as “package”
7. Click on Apply --🡪save

The above job will create an artifact from the code that was downloaded in stage 1. This artifact comes in the format of a war file

**Creating users in Jenkins:**

1. Open the dashboard of Jenkins
2. Click on manage Jenkins
3. Click on manage users
4. Click on create users
5. Enter user credentials

**Creating roles and assigning:**