**GIT: Version-controlled tool**

git is a version control tool used to keep track of the version of files and directories.

Some of the version control tools are GIT, SVN, CVS, GitLab, BITBUCKET, Azure Repo

Library/dependency/package – packagename –version

Git –version

Currently am using git 2.25.1 version.

How to initialize the git repo (how to convert the normal folder into git repo)

Switch into the folder and execute the command git init

How to configure the user

git config --global user.name “username”

git config –global user.email “mail id”

<https://github.com/sandy2498/scripts>

git token: ghp\_JkVHwwT3wO9p0FjhulQ2rDUob6hnEh2xNyW2

**git architecture there are 3phases.**

1. workspace: it is the place where we edit project-related files.
2. staging area: it is an intermediate area where we save changes.
3. Git repo: here versions of files will be tracked. Git commit command files will be moved to the git repo

Git clone url address

git status: it is used to check whether the files are in the workspace in the staging area or the get repo

git add --all: used to move all the files from the workspace to the staging area/ it will also move deleted files/changes

git add filename: used to move the specific file from the workspace to the staging area.

git add . : used to move all file from the workspace to the staging area.

git commit -m “message”: used to move the files from the staging area to the git repo

git push origin master: used to move the files from the local to the central repo

git log: it will display the history of a repo

git log filename : it will display the history of a specific file

git log -2 : it will display the history of the last 2 commits

git log commit id : it will display the history(username, servername, date, time and commit id) of the specific commit

**tag:** tag is a name given to set of versions of files and directories. tag easy to remember in future, it indicates milestone of a project

git tag: it will list all the tags created

git tag tagname: used to create a tag with updated version

git tag -d tagname : used to delete the tag locally

git push origin --delete tagname: to delete the tag in central repo through the terminal

git checkout tagname: it will switch (copy) to a specific tag

git push origin tagname: it will push from a specific tag

git push --tags:it will push all the tags to central repo

**Ass:**

1. create 3files and create a tag with tagname release1.0 and push this tag to the central repo.
2. modify above 3 files which are created
3. create a tag with release1.1 and push this tag to the central repo
4. create 2 more files and create a tag with release1.2 and push this tag to the central repo
5. 4 delete the tag release1.1 remotely (central repo)

**Branches:** branch is for parallel development, if two team/people will work on the same piece of code by creating different branches and we integrate by merging.

git branch : used to list all the branches

git branch branchname: used to create a branch

git branch -d branchname : used to delete the branch

git push origin --delete branchname : used to delete the branch in central repo

git branch branchname tagname/oldbranchname : used to create new branch from tag or other branch

git checkout branchname : used to switch to another branch

git checkout -b branchname : used to create a new branch and switch to that branch

git merge branchname : before executing this command, switch to the branch where we need to merge and type git merge branchname (give branch name from where we need to merge) command

git push origin --all : it will push all the branches to the central repo

git stash: when am working on one branch if I get any critical issue that needs to be fixed on another branch, I need to stash incomplete (half done) work on the current branch. Once I fixed the bug I switched back to the current branch. I will pop it and continue with the work.

**git stash**

Use: This will avoid committing the code on the wrong other branches.

git stash: it will stash the incomplete work

git stash pop : it will display stash entries

git stash --list : it will list the stash entries that you have

git stash --drop: it will remove the stash entries.

Merge conflicts: This will occur when the same line of code is modified on two different branches on the same file. When we try to merge these two branches, we get merger conflicts.

I don’t know whose changes should be considered and which changes should be considered I will contact the developer who modified the codes on two different branches, and they will discuss and give the new change. I will put this new change in the file and continue with the merge.

**Difference between rebase and merge:**

**Merge** just integrate the changes from one branch to another branch

**Rebase** is nothing but a merge, one branch will get added to the tip of another branch we can squash multiple commits as a single commit and then we can merge.

While merging it doesn’t allow any git function until unless merge conflicts resolved

Git rebase branchname

Git rebase -i HEAD~3

How do you merge specific change: **git cherry-pick commit id**

How do I know how many files are modified under one commit: **git show commit id**

How your going to check whether it is a good commit or bad commit: **git bisect**

How your going to check who modified each line of a code: **git blame**

**Git clone:** it is used to bring the central repo or remote repo to the local repo for the first time.

Syntax: git clone url

**Git pull:** it will bring the changes from the central/remote repo and merges to the local workspace automatically

Syntax: git pull url

**Git fetch :** it will bring the changes from central/remote repo and stores it in separate branch in local repo. We can review the changes and merge if it is required.

Syntax: git fetch url

**Difference between git and svn, CVS or (any other version control tools)**

Git is a distributed version control system the whole repo will be there in local workspace/local repo. If I want to go to the previous version of a code I can go in local workspace itself

Svn/cvs is a centralized version control system only latest version of code will be there in local workspace. If I want to go to the previous version of a code I need to checkout from central repo

Git has many advanced features like stash, reset and fetch whereas svn don’t have these advanced features.

**Repository:** where your Git projects are stored. All changes in a project and versions of saved files are in its repository.

Types of repo:

1. Bare repo : it acts as a remote repo we can only push and pull the code to this repo. We cannot run any git function

Syntax: git init --bare

1. Non bare repo: it is a local repo where we edit our project-related files, and we can run all git commands.

**Git revert:** used to undo the committed changes. History will be tracked.

**Git revert commit id :** it will revert the changes made in that commit

**Git revert HEAD~1:** it will revert the changes made in the last commit

**Git reset:** it is used to undo the committed changes but history will not be tracked. The name itself indicates reset HEAD to the previous commit.

**Types of reset:**

1. Mixed : it is used to move the files from the staging area back to the workspace

Syntax: git reset --mixed commit id

1. Soft : it will remove history and changes will be moved to the staging area

Syntax: git reset --soft commit id

1. Hard: it will remove the history and changes from everywhere (git repo, staging area, and workspace)

Syntax: git reset --hard commit id

.gitignore:

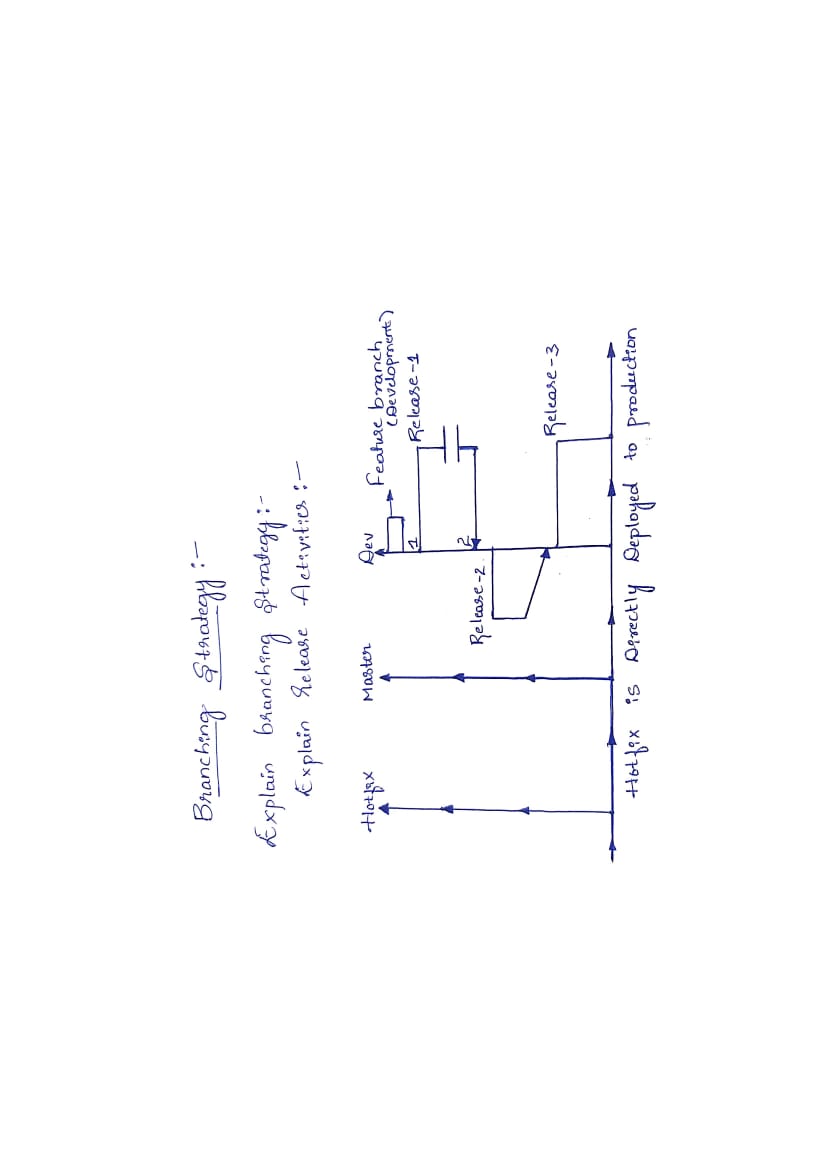
**How you’re going to give access to the repo:**

We need to click on the settings of the repo and click on collaboration and teams. Here we are going to add the users by adding a username. Only we will give read access.

**Branching strategy:**

Branches can be created for multiple reasons here we create branches for releases.

Development will be going on the dev branch once the code is ready for the first release on the dev branch, we create a separate branch for the first release, and we make a release from the release branch. Whatever issues related to the first release will be fixed on the first release branch. Parallel development will be going on the dev branch for the second release. Once the code is ready for the second release on the dev branch, before we create the release2 branch we merge the first release to the dev branch and then we create the release2 branch for the second release. Whatever the issues we have seen in the first release will not be seen in the second release.



**Feature branch:** developers will create their own feature branch and merge it with the dev branch.

**Release branch:** used to maintain the releases.

**Hotfix branch:** we used these branches to deliver a critical fix to the production environment.

**Master branch:** The developer directly can’t merge the code to the master branch, it is used to keep clean working code.

**Build:** binary or executable before testing.

**Release:** it is a tested build ready to release to a customer.

**Sanity or bvt (build verification test):** it is a basic functionality of a build that should never break.

**Hotfix or patch build:** it is a critical fix that needs to be delivered to a customer within a few hours. If I modify 5 files among thousands of files, only 5 files will be compiled and regenerated build. It takes less time.

**Load build (full build):** we compile source code from scratch. It takes more time.

**Release note:** it has two things tagname and known issues. DevOps engineer will prepare release notes.

**What kind of issues did you face while building? (build issues)**

Once the build is broken, we need to debug and identify why the build is broken it may be two reasons,

1. Compilation issue: If it is a compilation issue, we need to work with developers.
2. Build environment issue: If it is a build environment issue, we need to debug and fix it. This issue may be the reason for the following.
3. Memory is full on the server
4. CPU usage is high on the server
5. Slave machine will not be available in the Jenkins job
6. Compatibility issues (version mismatch)
7. Dependency not installed

**Build Tools:**

C and C++ : make => .exe , it will generate filename.exe

Java: maven, gradle, nodejs, sbt => it will generate .jar, .war and .ear

Dot net: npm, nuget

**Compiler**

C: Gcc

C++: g++

Java:javac

Note: we support deployment to the QA, dev, stable, UAT, and production. We deploy build to the respective server(environment) using the Jenkins job. To deploy to the QA and Dev environment we need manager approval.

**How are we deploying in the prod environment?**

Before deploying to the production environment, we need to test in lower environments (QA and Dev) if it is successful then we will raise a ticket to SRE (site reliability engineer) for deploying in a production environment. After approval with the help of the SRE team squad, we are going to deploy to the production environment.

**Maven repository:**

Maven search dependencies in the local repo, if it doesn’t find then it searches in the central repo (artifactory tools) if it doesn’t find then it searches in the remote repo.

Have you written pom.xml: pom.xml is given by the dev team. We mention the path for local dependency.

**Maven goal life cycle:**

It is a build tool used to run the java projects.

1. **Validate:** validate the project is correct and all necessary information is available
2. **Compile:** compile the source code of the project
3. **Test:**  test the compiled source code using a suitable unit testing framework.
4. **Package:**  take the compiled code and package it in its distributable format, such as a JAR/WAR
5. **Integration\_test:** used to do the integration test for the compiled source code.
6. **Verify:** verify the integration test to ensure the quality criteria are met
7. **Install:** install the package into the local repository
8. **Deploy:** we will deploy the final package into the artifactory tools.

**Tomcat server:**

Sudo apt update

Install java: sudo apt install java

Install tomcat: apt install tomcat

Public ip address:8080 (new tab)

C++ pearl java

Nexus

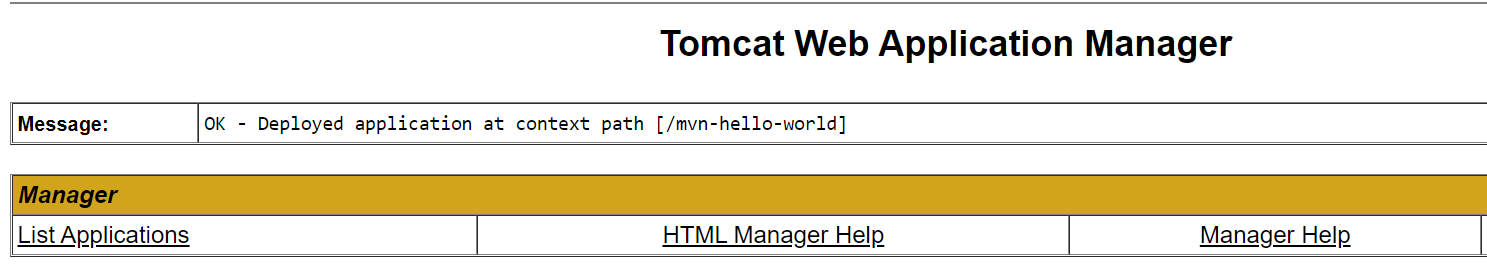
**For Tomcat installation please follow below link:**

**First, we need to install java**

1. sudo apt-get update or sudo apt update
2. sudo apt-get install default-jdk

[https://github.com/sandy2498/Simple-DevOps Project/blob/master/Tomcat/tomcat\_installation.MD](https://github.com/sandy2498/Simple-DevOps%20Project/blob/master/Tomcat/tomcat_installation.MD)

When we deploy we will get the below result in Tomcat.



**Where to add rules for a branch?**

We need to go for a setting in the repo, click on branches, and click on the add rule. Here we will add the rules whichever we require.

Continuous integration: integrate the changes from the development team continuously without manual intervention. We can give quick feedback to developers for their changes.

Continuous deployment: The tested build should be deployed to the production environment as early as possible.

Continuous delivery: each change from the dev team should be built and deployed to the test environment to test the changes.