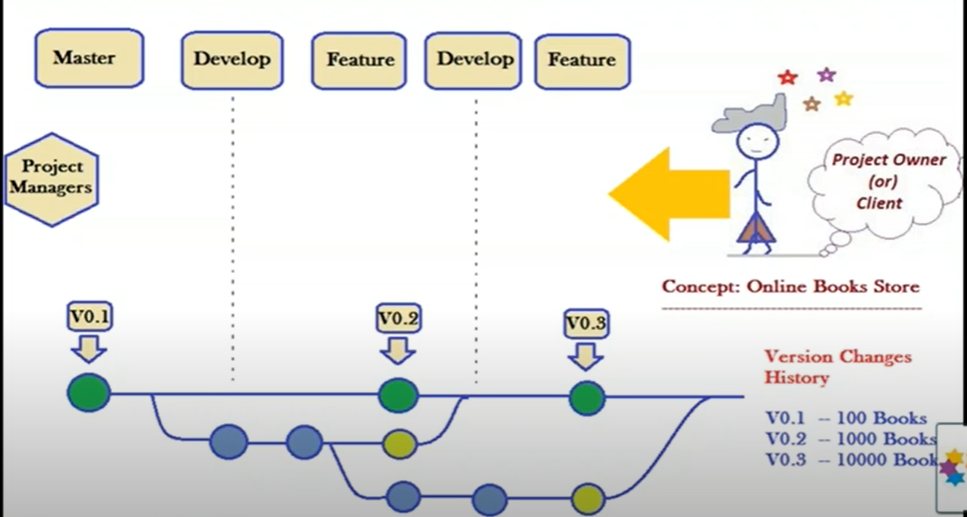
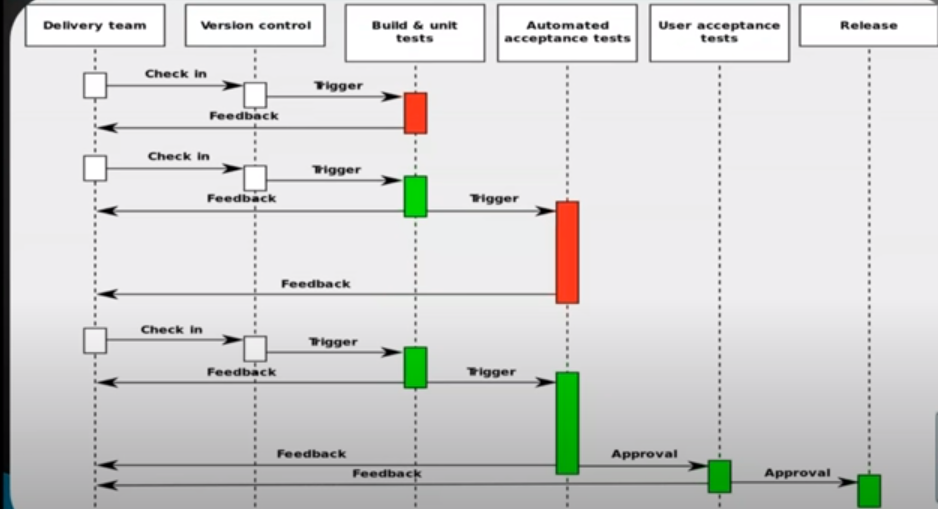
**What is VCS?**

* VCS is stand for Version Control System.
* It is a process to store all members changes centralized location and keep track all changes being done in the VCS by Whom, When and Why.

**(OR)**

* Version control, also known as source control, is the practice of tracking and managing changes to software code.
* Version control systems are software tools that help software teams manage changes to source code over time.
* 



VCS Tools:-

1. Other Tools : Perforce, CVS, TFS, Mercurial, etc..
2. Centralized VCS Tools : SVN, CVS
3. Distributed VCS Tools : Git, Bit Keeper, TFS, etc...

**What is GIT?**

* GIT is a distributed version control system
* SCM (Source code management) is used to track modifications to a source code repository

(OR)

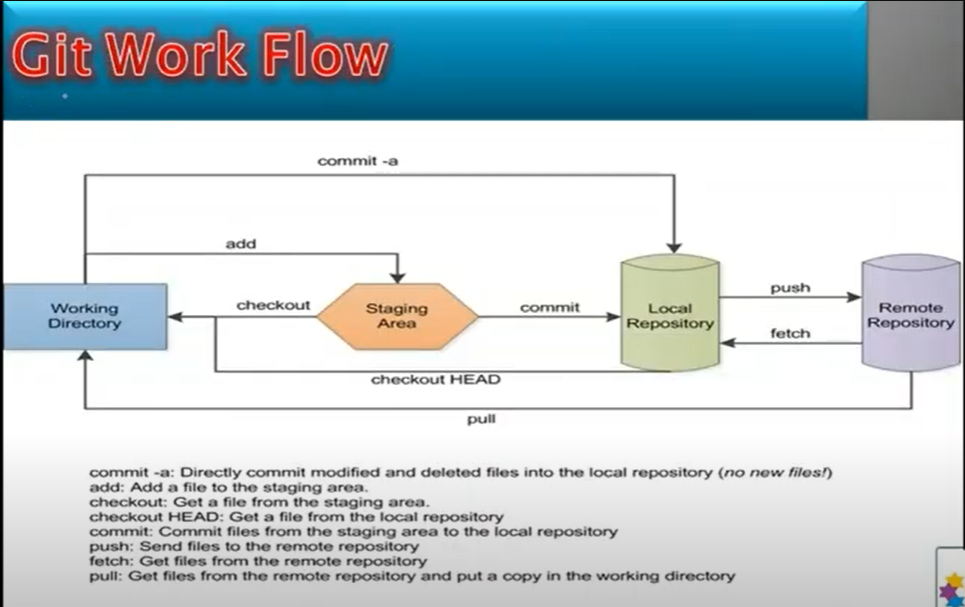
* **Git is open source and Source code management tool.**
* **It is a distributed version control system.**
* **It Support Offline work without server connectivity.**
* **It is providing code reusability and branching concepts.**
* **It is designed to handle everything from small to very large projects with speed and efficiency.**

**Features or (Advantages): -**

* + 1. Git is open source and free.
    2. Flexibility.
    3. Security
    4. Git is supported for all machines (Linux, Windows, mac, ect).
    5. It will support branching and merging.

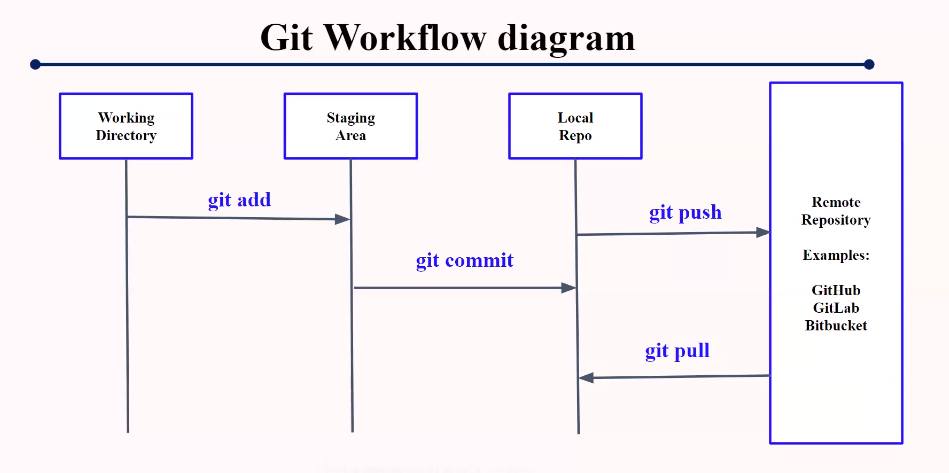
**What is the difference between Git and SVN?**

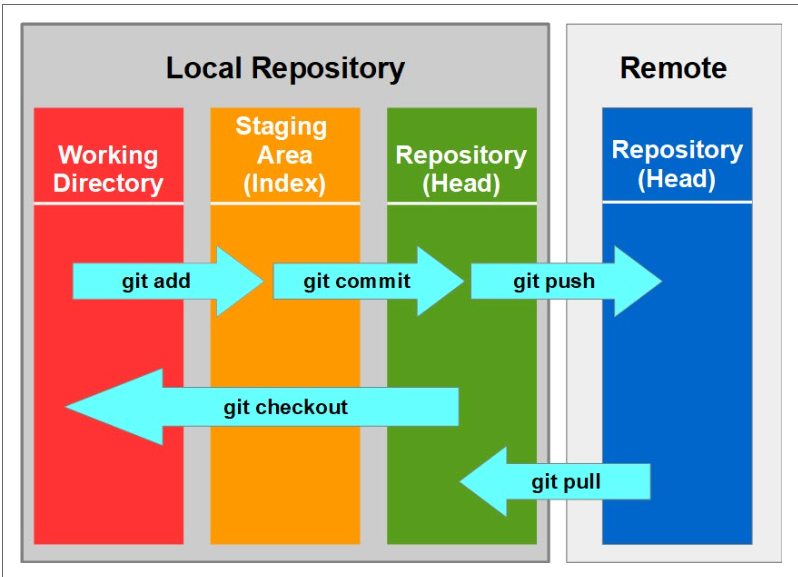
|  |  |
| --- | --- |
| **Git** | **SVN** |
| Git is a **Decentralized** Version Control tool | SVN is a **Centralized** Version Control tool |
| It belongs to the 3rd generation of Version Control tools | It belongs to the 2nd generation of Version Control tools |
| Clients can clone entire repositories on their local systems | Version history is stored on a server-side repository |
| Commits are possible even if offline | Only online commits are allowed |
| Push/pull operations are faster | Push/pull operations are slower |
| Works are shared automatically by commit | Nothing is shared automatically |



### What does Git do?

* Manage projects with **Repositories.**
* **Clone** a project to work on a local copy.
* Control and track changes with **Staging** and **Committing**
* **Branch** and **Merge** to allow for work on different parts and versions of a project.
* **Pull** the latest version of the project to a local copy.
* **Push** local updates to the main project.





A diagram of a computer

Description automatically generated

**Different B/W Git and GitHub.**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | | **Git** | **Git Hub** |
| 1 | Git is a software. | | Git is a Service. |
| 2 | Git is a command-line tool | | GitHub is a graphical user interface |
| 3 | Git is installed locally on the system and maintained by Linux. | | GitHub is hosted on the web, and it maintained by Microsoft. |
| 4 | Git is focused on version control and code sharing. | | GitHub is focused on centralized source code hosting. |
| 5 | Its desktop interface is named Git Gui. | | Its desktop interface is named GitHub Desktop. |

**Different B/W GitHub and GitLab.**

|  |  |  |
| --- | --- | --- |
| **S.No** | **GitHub** | **GitLab** |
| 1 | It is not open source. | It is open source for the community edition. |
| 2 | GitHub is written in Ruby. | GitLab is written in Go, Ruby, and Vue.jst. |
| 3 | Users can have an unlimited free repository. | Users can make a public repository. |
| 4 | It allows users to navigate usability. | GitLab provides the navigation into the repository feature. |

**Git command’s: -**

**File’s created command – $ Touch <File’s name’s>**

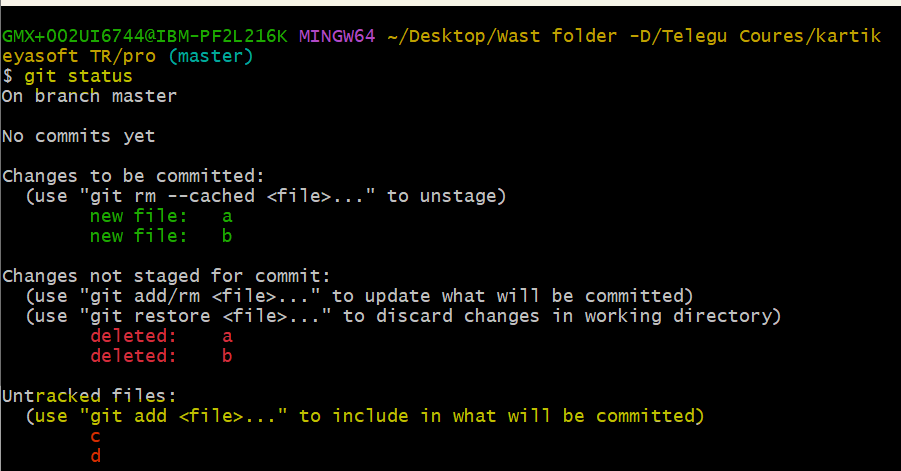
1. **Gig init: -** This command is used to start a new local repository.

🡺Usage: git init [repository name]



1. **Git status:** - Lists all new or modified files to be committed.

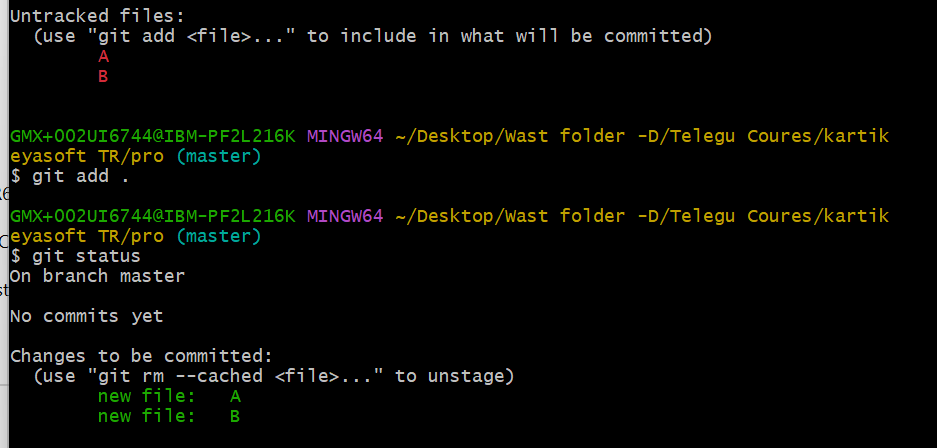
🡺Usage: git status



🡺**Untracked files** are files that have been created within your repo's working directory but have not yet been added to the repository's tracking index using the git add command.

**.git files:** - It a Locally repository

3.**Git add . :-** Prepares the file for commit by logically moving it to the staged area

****

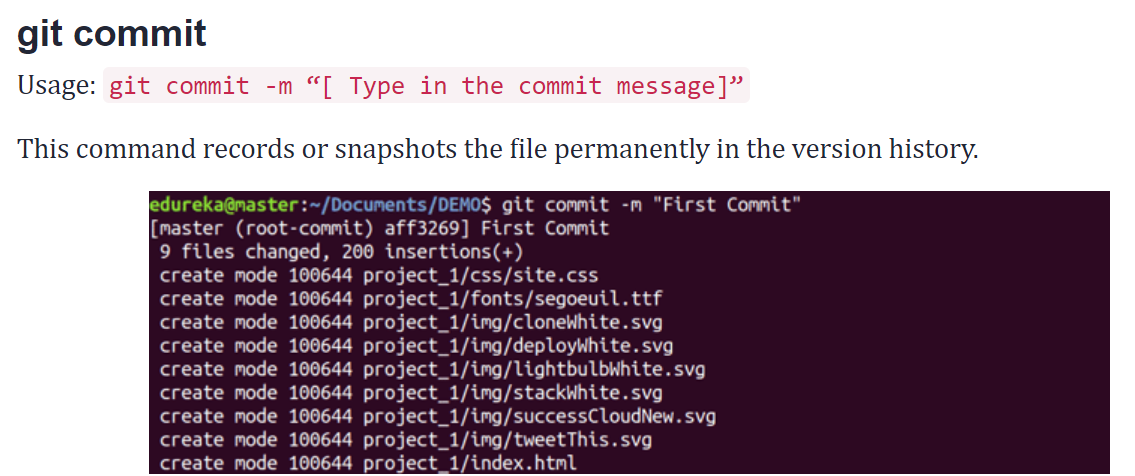
**A diagram of a diagram

Description automatically generated**

**4.git commit -m<message>**

**🡺**staging area to local Repo we will used git commit -m

**🡺** Adds the staged files permanently in version history.

****

**5.git log** : - Lists version history for the current branch and it generated one commit id.

**A screenshot of a computer program

Description automatically generated**

**Created local Remote repo?**

**6.Git init –bare <directory>**

**🡺**it will create a local remoted repo.

🡺 A bare git repository is intended to be used as a remote repository where code is shared between members of the team.

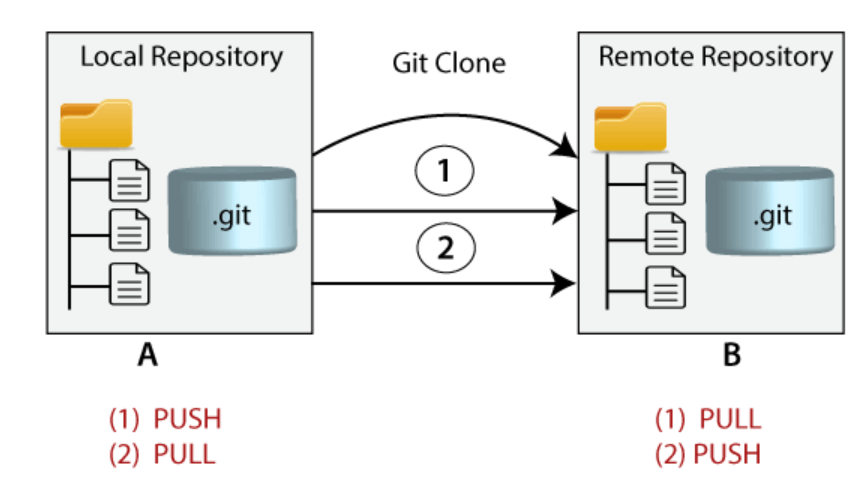
**Different b/w git init and git init –bare**

**Git init 🡪** It created a new local repository and .git file.

**Git init –bare 🡪** It created a remote repository.

**7.git clone**

**🡺** The **git clone** is a command-line utility which is used to make a local copy of a remote repository. It accesses the repository through a remote URL.

****

**8. git pull: -** Git Pull is a command used to update the local version of a repository from a remote repository. It is a mixture of two other commands:

1. git fetch.

2. git merge

(OR)

* The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content.

**9.git push: -**

🡺 The git push command is used to upload local repository content to a remote repository.

A diagram of a diagram

Description automatically generated

**10. git config: -**

* **git config** command is a convenience function that is used to set **Git configuration** values on a global or local project level.
* **Configure the Git for the First use.**

1To register a username, run the below command:

* $ git config --global user.name “File name”

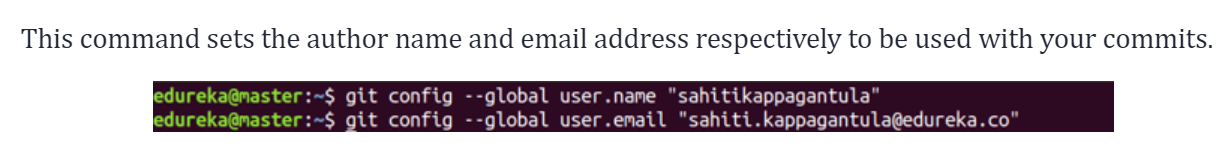
2.To register an email address for the given author, run the below command:

* $ git config --global user.email "File name"

3.global we need to push

🡺 $ git config –global push. default simple

4.git list 🡺q mean quite that file’s



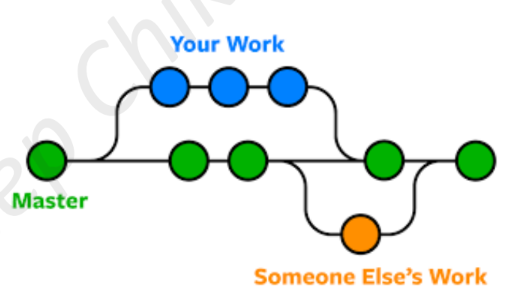
**why we are using . gitignore file?**

🡺The purpose of gitignore files is to ensure **that certain files not tracked by Git remain untracked.**

🡺To stop tracking a file that is currently tracked, use **git rm --cached** to remove the file from the index. The filename can then be added to the .gitignore file to stop the file from being reintroduced in later commits.

**11.Git Branch**

* A branch represents an independent line of development.
* The git branch command lets you create, list, rename, and delete branches.
* The default branch name in Git is master.



**Note:-** If you perform git branch first time, you can’t see any branch including master also. You can see when you commit something first time to repo. Otherwise, master is not visible

* To Creating a new branch ⟶ **git branch <branch\_name>**
* To see current branch ⟶ **git branch**
* To switch one branches to other branch ⟶ **git checkout branch**
* To create and switch branch at a time ⟶ git checkout -b branch\_name
* To rename a branch ⟶ git branch -m old\_name new\_name
* To delete a branch ⟶ **git branch -d branch\_name**
* To Rename the current branch ⟶ git branch -M newBranchName
* The -d option will delete the branch only if it has already been pushed and merged with the remote branch.
* To delete a branch forcefully ⟶ git branch -D branch\_name
* Use -D instead if you want to force the branch to be deleted, even if it hasn’t been pushed or merged yet. The branch is now deleted locally.
* **git branch --merged**

**Note:-** If you create one file inside a branch from master. It will be present in all branches & master. But, whenever you are committing the file into a particular branch. It will not present

**Feature Branch**

* They are short lived branches.
* We can simply called ephemeral i.e. short lived objects
* we delete it after it is integrated to develop

12 **Git merge: -**

* Getting all commits from one branch to another branch.
* Means getting files and commits from one branch to other.

**Note:-** First, which branch you need to merge, go inside that branch and perform that merging

Eg: so, I’m going to master i.e.., git checkout master then you can do merge.

* **git merge <branch\_name>**
* Cancelling the merge ⟶ git merge –abort

13 **git rm <file name>**

* Deletes the file from the working directory and the staging area
* git rm -rf

**14 git Stash 🡪** **It moves to back staging area to working DIR.**

1. Using the git stash command, developers can temporarily save changes made in the working directory.
2. It allows them to **quickly switch contexts** when they are not quite ready to commit changes.
3. And it allows them to switch between branches more easily.
4. It saves changes in working and index areas and saves it to a different location and making a way for other important tasks.
5. It can be access only inside the branch.
6. Generally, the stash’s meaning is “store something safely in a hidden place”.

* Apply the stash ⟶ **git stash (or) git stash apply**
* Remove the stash ⟶ **git stash clear**
* Check the stashes ⟶ **git stash list**
* Remove the last stash ⟶ git stash pop
* Remove particular stash ⟶ **git stash drop stashID**

eg:(git stash drop stash@{0})

**15 git diff**

* git diff lists out**the changes between your current working directory and your staging area.**

1. **git diff head** 🡪local repo changes

* The **git diff HEAD [filename] command** allows you to compare the file version in your working directory with the file version last committed in your remote repository. The HEAD in the git command refers to the remote repository.

1. **Git diff --cached** 🡪 only staging area changes.

* will show you the difference between the index and your last commit; what you would be committing if you run "git commit" without the "-a" option.

**16 git reset**

* reset is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit.
* These forms are as follows.

1. **git reset --soft -** this will keep your files, and stage all changes back automatically.[ Git reset soft undoes the changes to the index]
2. **git reset --hard -** which will remove any changes and remove them from the local directory. [ Git reset hard undoes changes and removes the associated files from the working directory, staging area, and commit history]
3. **git reset –mixed -** This keeps all files the same but unstages the changes.[  Git reset mixed untracks files without deleting them from the working area.]

**git revert <commit id>**

**git rebased**

* **git rebased -i HEAD~8**

**git tag**

**git show --stat**

<https://dzone.com/articles/top-20-git-commands-with-examples>

<https://www.devopsuniversity.org/git-tutorials-git-commands/>

https://5minslearn.gogosoon.com/blogs

Step 1: First created file’s Using command – $ Touch <File’s name>

Step 2: git init (used to start a new local repository or New initialized status)

Step 3: git status (Lists all new or modified files to be committed or Untracked files display)

Step 4: git add . ()

**Build tools. 🡺 Maven, Ant ect**

* **Package**
* **Repo**

1. **Local repo**
2. **Remota repo**

**🡺git hub**

**🡺git lab**

**🡺Bit bucket**

**CI/CD 🡺 devops**

VCS -- DEV(Change) --- Build tool – maven (it is user for java project)

1. C, C++ 🡺 Make file.
2. .net 🡺 Visual Studio
3. Java 🡺 Maven, Gradle, Ant

Old day’s how there are usering.

* Dev – code -- .xml --- Java.jar<file-name>

Build process 🡪 Artifact (Package Jar, war, ear)

**Maven Installation**

**1.windows**

**2.Linux 🡪 maven .tar file**

**JDK**

**Java Home path**

**Download Maven Zip**

**Unzip maven file**

**Maven home path**

Maven

=======

Build tools ------Maven , Gradle, Ant

**Ant** 🡪 It stands for Another neat Tool.

🡪 It was the first build tool, but it is complicated because of large XML configuration file.

**Maven & Gradle** 🡪 Needs less configuration but follows more conventions.

🡺Pom.xml

POM stands for **"Project Object Model".**

[**https://www.youtube.com/watch?v=MPfRnyZ5G6U&list=PLpLBSl8eY8jTWanMqhDdMtQsvOYA28kaF**](https://www.youtube.com/watch?v=MPfRnyZ5G6U&list=PLpLBSl8eY8jTWanMqhDdMtQsvOYA28kaF)

[**https://www.simplilearn.com/tutorials/maven-tutorial/introduction-to-maven**](https://www.simplilearn.com/tutorials/maven-tutorial/introduction-to-maven)

**why do we use Maven?**

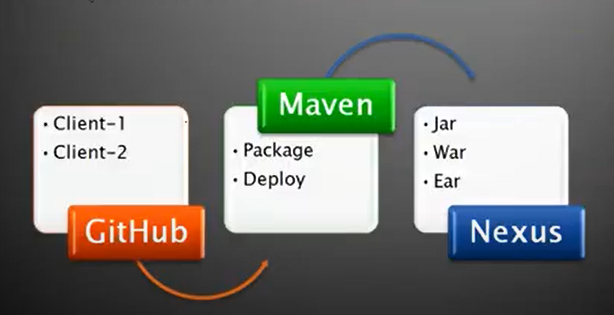
**🡺The tool is used to build and manage any java – based project.**

* **It helps in getting the right JAR file for each project dependencies.**

**Build Tools**

* The Build tool is automated the creation of executable applications from source code.
* It reduces the normal project deployment time.
* It releases executable files when the time of deployment in various formats -**Jar, War, Ear...** depending on project life cycle.
* Build tools are --> **Maven, Ant, Gradle,** Apache Builder, Visual Builder.

**Maven Working: -**



**Day-15 (Class Note’s)**

* mvn archetype: generate
* mvn archetype:generate > (Test name)

**MVN Goal’s**

* mvn validate
* mvn compile
* mvn test
* mvn package (compiler and test it will contend one single package)
* mvn install
* mvn deploy

Tomcat download

**Maven project co- ordinates**

1. Groupid
2. Artifact id
3. Version
4. Packaging

Group-id: -

* A Unique id used to identify the project of a client.

Ex: - I’m working in Infosys and Im creating project for the client version, Now I will use group id for the project as com.verizon

Artifact-ID – project name/ final war/ jar name

Version – version of the project

Packaging – jar/ war