

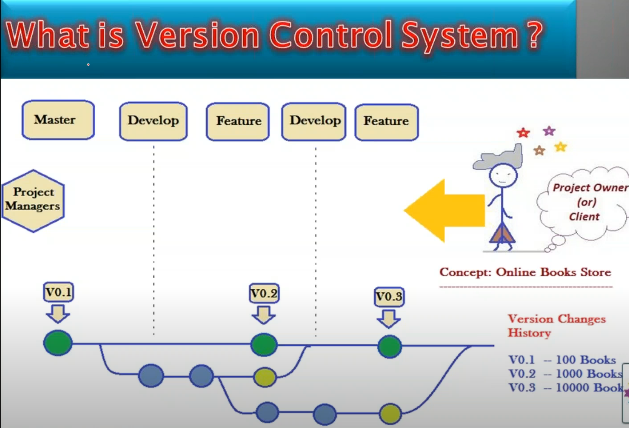
**Source code Management System(SMS) Or Version Control System(VCS):**

Real life projects generally have multiple developers working in parallel. So they need a version control system like Git to make sure that there are no code conflicts between them.

Also, the requirements in such projects change often. So a version control system allows developers to revert and go back to an older version of their code.

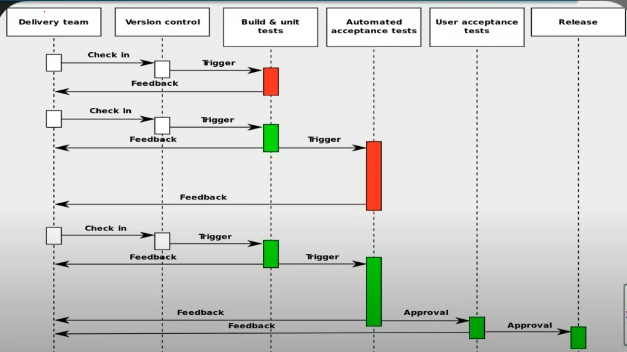
In real time working environment so many devolopers works on the code devlopment,if any thing goes wrong with code and conflict issues araised at testing,deployment stage etc,at this scenarios we can easily track the all details with this version control system tools.

**Version control system with Sample Example:**



* In this Example Client want to start Online Book Store so he approaches one of the IT company to start their project.
* Initially he want to sell 100 books through website.so based on his requirement developers write code(V0.1) and sucessfully deployed.
* After some days client want to sell 1000 books.so again developers need to develop the code.but in this case developers no need to start the work from the stratch level.They will develop (V0.2) with the help of (V0.1)
* Suppose client changes his decision and want the previous vesrsion.Then devlopers can easily restore the previous version with the help of (V0.1)
* Like this (V0.3=V0.1+V0.2)
* All these versions are maintained by using snapshots
* So all these activities done by using version control system tools.

**VERSION CONTROL SYSTEM WORK FLOW:**



* Delivery team always checks that whether the product is ready or not when it will need to be release.
* In this checkin process the 1st phase will be version control,here they check the developers code. and passes to other phase tests.code need to be pass in all these phases tests to release.if it fails in any stage it will directly revert back to version control system.
* So if the product need to be build,test,deploy,release etc,it would be properly done in initial stage which is Version control system.
* **VCS DEFINATION:**

VCS is a procss to store all members changes in centralized location and keep track all changes

being done in the VCS by whom,when and why.



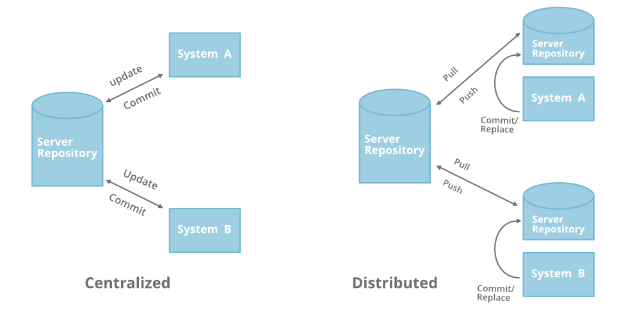
**VCS TOOLS:**

****

**GIT**

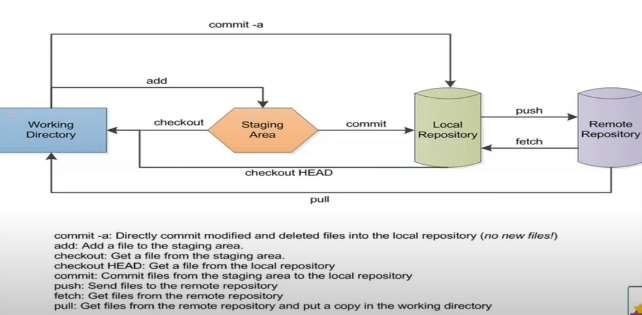
* Git is currently the most popular implementation of a distributed version control system.
* Git originates from the Linux kernel development and was founded in 2005 by Linus Torvalds.
* It is decentralized VCS Tool.
* It supports offline work without server connectivity.
* It is providing code reusebility and branching concepts.

**Centralized VCS (vs) Distributed VCS:**

****

* Performance of distributed systems is better, because there is no waiting for locks to happen across potentially slow network connections. Also, the complete code base is already on your local system.
* Branching and merging is much easier to achieve in a distributed system, largely because it’s built in to the way the system works.
* With a distributed system, you don’t need to be connected to the network all the time.

**GIT WORK FLOW:**

****

**WORKING TREE:**

The Working Tree is the area where you are currently working. It is where your files live. This area is also known as the “untracked” area of git. Any changes to files will be marked and seen in the Working Tree. Here if you make changes and do not explicitly save them to git, you will lose the changes made to your files. This loss of changes occurs because git is not aware of the files or changes in the Working Tree until you tell it to pay attention to them.

Run the command **git status**. This command will show you two things: The files in your Working Tree and the files in your Staging Area.

**THE STAGING AREA(INDEX):**

The Staging Area is when git starts tracking and saving changes that occur in files. These saved changes reflect in the **.git directory**.

**LOCAL REPOSITORY:**

The Local Repository is everything in your .git directory. Mainly what you will see in your Local Repository are all of your commits. It is the area that saves everything (so don’t delete it). That’s it.

The git command git commit takes all changes in the Staging Area, wraps them together and puts them in your Local Repository. A commit is simply a checkpoint telling git to track all changes that have occurred up to this point using our last commit as a comparison. After committing, your Staging Area will be empty.

**BRANCHING:**

Branch operation allows creating another line of development.To differentiate and maintaining privacy between the developers we are using brancing concept.

For example, we released a product for V.1 version and we might want to create a branch so that the development of V.1 features can be kept separate from V.0 bug fixes.

Generally Branches are used in project to

\*work with new modules without disturbing the main project

\*the default git branch is **master**

**\***we use different branches to develop new modules

**\***it would be merged with **master** later

**Note:** In any organzation nobody don't commit changes on master branch.that is the reason of creating new branches.developer works on different branches,when code get succeed on different phases of tests then brances will merge into master branch.

Simply we can say that, what was deployed in prodction environment that is master branch code.

**MERGING:**

\*The **git merge** command integrates the independent lines of development into a single branch.

\*Isolating **features**into different branches is a crucial practice for any serious developer. By separating each feature, bugfix or working experiment you will avoid a lot of problems and keep your development branches clean.

At some point, a piece of code will reach a state where you’ll want to integrate it with the rest of the project. This is where the git merge command comes in.

\*Whenever we are doing merge operation we have to give reason for that.generally we gives the details that,what modifications done,

**Preparing to Merge**

Let’s assume you want to merge branch **branch1** into your **master** branch.

Before you start, how to make sure that you are ready to merge changes?

* Check if your local repository is up to date with the latest changes from your remote server with a **git fetch** .
* Once the fetch is completed **git checkout master.**
* Ensure the **master** branch has the latest updates by executing **git pull.**
* Checkout to the master branch that should receive the changes, in our case that is master.

**Note:** When we merge other brances to master then we are switched to themaster first.

**HEAD:** representing your current working directory, the HEAD pointer can be moved to different branches, tags, or commits when using git checkout

**GITHUB:**

\*GITHUB is a Web-based Git version control repository hosting service.

\*It provides all of the distributed version control and source code management (SCM) functionalities of Git while topping it with a few of its own features.

\*It is a heaven for the developers where they can store their projects, and get connected with like-minded people.

\*You can think it as “cloud for codes”.

\*It works along with the collaboration of Git. While Git is a command line tool, GitHub is a Web-based graphical interface which provides you with the access control, basic task management tools along with several collaboration features. Moreover, GitHub can host your project’s source codes in various programming languages and keep the track of the each of the changes made to every iteration.

**GIT&GIT HUB COMMANDS:**

**git status -** it allows you to get a comprehensive list of what is in you Working Tree and Staging Area. It lets you to know what files have been added, modified, and which files git is tracking.

**git diff -** it alows you to compare the diff versions of your files.

The most common scenario is use "diff" is to see what diifferences after your last commit.

It is only for the continous modifications before commit it.git diff is doesn't work for newely created files.

**git stash** - it is used to create workspace for the incomplete statements

**git stash list,git show stash id** - to get the stash id

**git stash show -p stashid** -to show stash file details

**git stash pop stash id** - to get back the file to working directory

**git ignore** - it is used to create work space for the incomplete statements which are not accessed long time

git ignore is only work s for newely created files

if we don't want to track the newly created file for long time then we created .ignore file and we put that file insides the **.ignore file**

**git reset - soft -** To delete only comments (or)statements.

**git reset** --hard -To delete history as well as file

**git revert -** To correct the wrong operations **.**

**git fork -** to create a copy of the main repository of a project source code to your own GitHub profile.

**git init** - Create a new git repository

**git add newfile**-Add a new file to your staging area

**git commit** - Adds staged changes to your local repository

**git push remote branch**- Push local repository changes to your global repository

**git pull remote branch**-pull code from your global repsitort to your local repository

**git branch** -See local branches

**git branch newName** -Create new local branch

**git checkout branchName**-Switch branches

**git branch -d branchName-**deleting branches

**git merge** -to merge the branches

**git branch --merged-** to get details of merged branches

**git branch --no -merged** - to get details of non merged branches

**git log** -Show recent commit history

**git show commit\_id** -show details of specific commit

**git help** -manpages for git

**git help gitCommand** -man pages for specific git command