**Git & GitHub**

Version Control:

* Management system that manages the changes of the documents, computer programs, large web sites & other collection of information to recall specific versions later.
* The changes are termed as versions.
* It creates snap shot of entire project & saves it.
* These snap shots are called versions.
* Version Control System (VCS) stores different versions (previous) in a repository and can be recalled from the same. There are Local, Central and Distributed Version Control systems.

Why Version Control?

* For Collaboration:
  + It provides shared work space & tells what changes taken up and real time updates. Collaboration between all the developers.
* For Managing Versions:
  + All the changes that are made to the file will be tracked.
* Roll back functionality:
  + It is easy to roll back to previous version if any issue in the current version.
* Reducing Downtime:
  + To reverse the faulty update & save time.
* Analyze & Compare Versions:
  + To analyze and compare versions.
* Snapshots of all versions are properly documents & stored.
* Versions are also named accurately.
* A backup is always available in your local servers.

Version Control helps to analyze the project

* Version Control System (VCS) provides proper description when the version changes.
* What exactly was changed? When it was changed.
* So by above reasons, one can analyze how the project evolved between versions.

Version Control Tools;

* Git (Distributed VCS)
* Apache Sub Version (SVN). It is a centralized version control system.
* Concurrent Version System (CVS). It is Distributed VCS.
* Mercurial (Distributed VCS).

Repository:

A repository is a directory or storage space where projects can live. It can be local to a folder on computer or it can be storage space on another online host (Github). Can keep code files, text files, image files in repository

Difference between Centralized and Distributed Version Control System (VCS):

|  |  |
| --- | --- |
| **Centralized VCS** | **Distributed Version VCS** |
| It consists of central server where the source code is managed. Each developer connects to the central server and updates their specific code to the central server | Each developer has a local copy in their local repository |
| Any changes made to a code are committed to the central repository | Any changes made to a code are committed to the local repository |

What is Git?

* It is distributed version control tool that supports distributed non-linear work flows by providing data assurance for developing quality software. It lets a developer and team of developers work together on the same project from anywhere. Team members can work on files and easily merge their changes into one source.

Git & GitHub:

* Git is a local repository and VCS tool (Version Control Management Tool).
* GitHub is central repository. It is code hosting platform for version control collaboration.
* GitHub is a company to host central repository in a remote server. It is similar to social network.

Features of Git:

1. It is Distributed & it is Compatible
2. It supports Non-linear or parallel development.
3. It takes the snapshots & records changes made to a file rather than file itself. It is Branching
4. It is Light weight
5. It is Speedy. Every change that made to the specific code is lightning fast compared to other VCSs.
6. Every task in Git is undo-able so Robustness.
7. It is reliable & integrity. No changes can be made without Git recording it.
8. It supports Branching or paralleling. Every collaborator’s working directory is a branch by itself.
9. It is Secure.
10. It is Economical. It is released under GPL’s license. It is free and open source.

Distribution Feature:

* It allows distributed development of code.
* Every developer has local copy of entire development hosting and changes are copied from one repository to another.

Compatible Feature:

* Compatible with existing systems (VCS) & protocols.
* SVN & SVK repositories can be directly accessed using Git-SVN.

Non-Linear Feature:

* It supports non-linear development of software.
* Includes various techniques to navigate & visualize non-linear development history.
* Git records current status of project by creating Tree graph from the index.
* DAG – Directed Acyclic Graph.
* Non-Linearity can be made by branching.

Branching Feature:

* Branching helps you to make non-linear software development.
* Includes various techniques to navigate & visualize non-linear development history.
* Git has branching model feature.
* Master branch.

Light Weight Feature:

* Uses lossless compression technique to compress data on client’s side.

Speed Feature:

* Fetching data from local repository is 100 times faster than remoter repository.
* Git is one order of magnitude faster than other VCS tools.
* It is 10 times faster than other VCS tools.
* It is developed in C language, since close to machine language.

Open Source Feature:

* It is open source, created by Linus Torvalds.
* Can modify its source code according to the needs

Reliable Feature:

* On events of system crash, the lost data can be easily recovered from any of the local repositories of the collaborators.

Source Feature:

* Git uses SHA1 to name and identify objects.
* Every file and commit is check summed and is retrieved by its checksum at time of check out.
* SHA1 (Secure Hash Algorithm) cryptographic algorithm converts commit object into photo digital hexadecimal code.
* SHA1 uses MD-4 & MD-5 techniques & algorithms.

Economical Feature;

* Released under GPL’s license. It is free (limited). GPL means General Public License.
* All heavy lifting is done on client side; hence a list of many can be saved on costly servers.

What is Repository?

* A directory or storage space where projects can live.
* It can be local to a folder on computer or it can be space on GitHub or another online host.
* Can keep files, text files, image files, name it, inside a repository.
* There are two types of repositories:
  1. Central Repository
  2. Local Repository
* Central Repository:
  1. It is typically located on remote server.
  2. It exclusively consists of “Git Repository” folder.
  3. It is meant for team to share and exchange data.
* Local Repository:
  1. It is typically located on local machine.
  2. It resides as a “Git Folder” inside projects root.
  3. Only admin of the machine can work with this repository.

Creating Central Repository on GitHub:

1. Link : github.com
2. Install Git on local machine.
3. Create an account in GitHub.

Git operations & Commands:

1. git init: To create new local repository with a folder name “**. git**”
2. git clone: Download or create a copy of original repository from remote GitHub

to local machine.

1. git fork: Creates a copy of the original remote repository into GitHub account.

Syncing Repository:

* **git add** origin <link> : to add remote repo.
* **git pull** : to pull files from remote repo.
* **git push** : to push changes into remote repo.
* **git remote** add origin <url> : (url can be copied from github account)
* Making Changes:
  1. Commands:
     + git status: tells which files are added to index & are ready to commit.
     + git add : lets add files to index
     + git commit : it refers to recording snap shots of the repository at a given

time. Committed snapshots will never change unless done explicitly.

* git add <name of the file> : to add files to index
* git commit –m : to see the commits by all the collaborators
* git commit –m <“add messages”>: to commit all the files at once.
* git log: to see the commit objects & messages.
* git add –A: to add multiple files.

Parallel Development Branching:

* Branches are pointers to a specific commit.
* Branches are of two types:
  1. Local branches
  2. Remote tracking branches
* Remote branches connect local repo to central repo.

Creating Branches:

* Commands:
  + git branch <branch name> : contains all branches that originate from master

branch

* + git checkout <branch name>: to switch from one branch to other.
  + ls: to list all the files in a branch/master.

Merging:

* It is a way to combine the work of different branches together (into master branch).
* Allows branching off, developing a new feature & combining it back in (master branch).

Merging Branches:

* git merge <branch name>: branch is source branch/branch, not master.
* cat: to see the text in a file (as in Linux).

Rebasing:

* This is also another way of merging.
* This is also a way of combining the work between different branches.
* This can be used to make a linear sequence of commits.
* Also to integrate changes made in one to other branch.
* This places entire commit history of branch on the tip of the master.
* Used for much cleaner history & also to reduce branches
* Commands;
  + git rebase master (master branch name): to contribute all the files to central repo.
* Generate ssh key in order to connect through ssh to central repo.
* Command:
  + ssh –keygen : to generate ssh key
  + cat <ssh url> : to see public ssh generated key
  + ssh –t [git@github.com](mailto:git@github.com) : for authentication
  + git push <remote name> <local branch name> : pushes all the work of local repo

branch into central repo branch

* + git push <remote master name> : to add all the files of branch of central repo

to master of central repo

1. Initialisation:
   1. git init
2. Creating global parameters:
   1. git config - -global user.name “name”
   2. git config - -global user.email “email address”
3. Status finding:
   1. **git status** command is used to display the state of the repository and staging area. It allows us to see the tracked, untracked files and changes. This command will not show any commit records or information.
   2. To find the status of the repository(local)
   3. Master branch is the default branch when a local repository is initialized
4. Git works in three logical areas:
   1. Working Directory : All edited files lies in working directory
   2. Staging area: **git add** command is used to move the application code file in working directory to staging area.
      1. Command: **git add file name**(with extension)
   3. Local repository: **git commit** command is used to commit the application code

file to the local repository.

* + 1. Command: **git commit –m “message”** (for further understanding why the file was committed)

1. Command: **git log** is used to track the changes what have been made to the local repository
   1. git log command will show the # id for specific commit and the author who has made that particular commit
2. Command: **git show** is used to check the changes made with the git commit command
   1. **git show commit id**(at least 7 to 8 characters)
3. Command: git ls-files is used to list out all the files that are tracked by the local repository
   1. **git ls-files**
4. Command: touch command is used to create a file without any content. The file created using touch command is empty.
   1. **touch file name**
5. C

***Git & Git Hub from DevOps tutorial @ ADAM***

Agile SLDC is a continuous development such as:

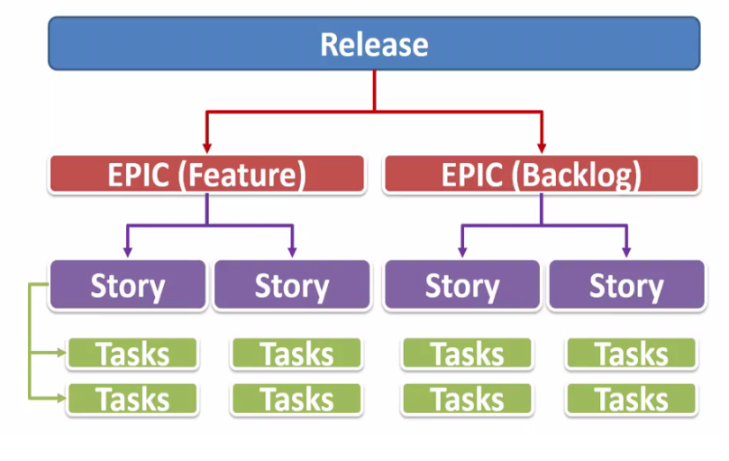
* Release Planning
* Agile – Scrum – Sprint - Planning
* Setup Distributed Version Control System
* Define Developer Work flow policies
* Branching strategy
* Dockerize the components/product
* Tools : Git - GitHub – Gitlab – JIRA - Dockerfile – JBOSS

Release Planning:

* List all the requirements(New Functionality, Enhancement, Bug fix)
* Bug tracking tool: JIRA, Bugzilla
  + User story / Unique ID
  + Priority(Highest, High, Medium, Low)
  + Timeline / Owner
  + Tasks(Unique ID)

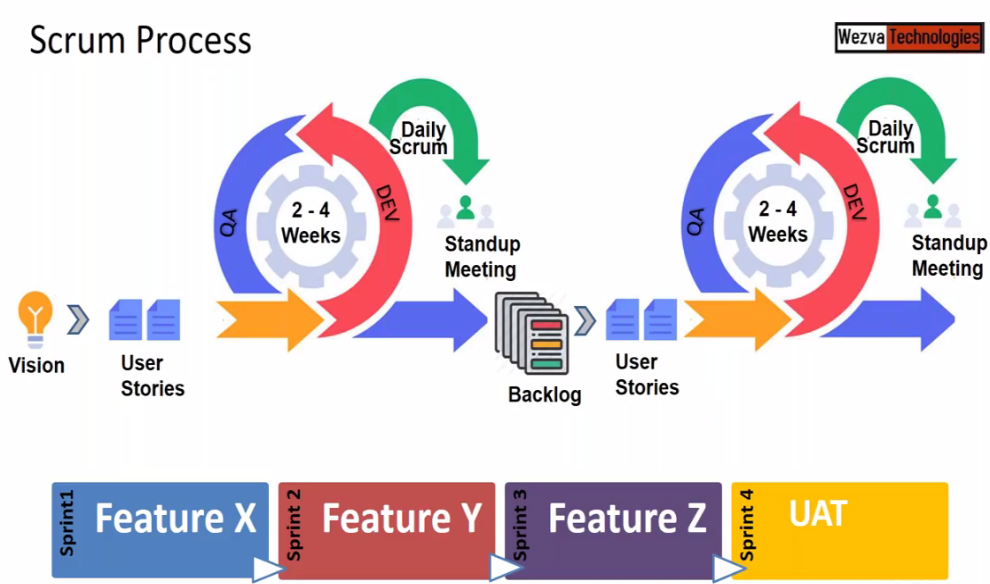
Agile:

* Agile project management is a project philosophy or framework that takes an iterative approach towards the completion of a project.
* Agile is a philosophy or orientation
* Split into module or components
* Sprints
* In Agile there are two ways to approach SDLC. Module by module development – Coding & Testing
  + Scrum: Dev / QA Team
  + Kanban: Operations Team
* In Agile all the requirements will be split into EPICs’. Every release in Agile methodology will be split into EPICs’
* EPIC is collection of some functionalities that need to work
* It can be new functionality to add or a functionality that need to add now to the previous version
* Every release tab is divided as EPIC(Feature) or EPIC(Backlog)
* EPIC will be further divided into User stories
* Every story is further divided into Tasks.



**Scrum:**

* Scrum is a type of framework for implementing the Agile methodology to complete software development projects.
* The Scrum framework operates as an iterative software design that allows project managers to establish team roles, responsibilities, collaborative meeting schedules and required tools.
* Scrum meetings to collaborate on progress and performance support productivity and successful outcomes.
* Run a Scrum to define the specifications for each project you initiate with your team, and then break the entire Scrum process down into sprints.



* Scrum process is the continuous process of developing, testing & review
* The user stories that has not been completed within that sprint period will become backlogs and becomes user stories along with the new user stories for the next sprint as shown in the above figure.
* Every day there will be review on the tasks that had been done is called Standup meeting or Scrum meeting
* Scrum team consists of
  + Product owner
  + Scrum masters
  + Cross functional team members or Regular engineers.
  + The product owner is responsible for maximizing the value of the product,
  + While the Scrum master is accountable for ensuring that the project team follows the Scrum methodology.

**Sprint:**

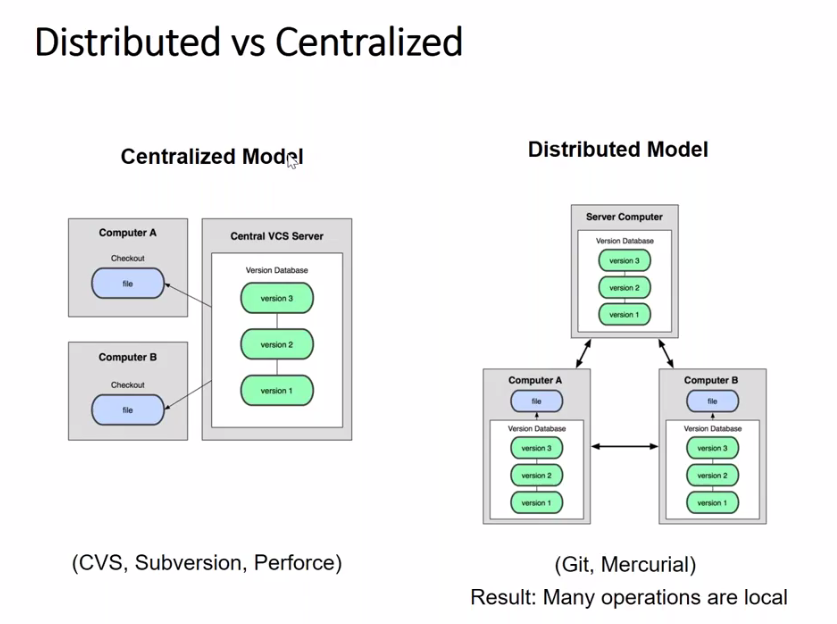
* A sprint is a process within the Scrum framework that divides project workflow into time-box iterations.
* Sprints typically take about two weeks to complete.
* When a sprint is currently in progress, development teams collaborate during the Scrum meeting to track progress and integrate modifications if improvements are necessary.
* At the end of a sprint, development teams conduct a sprint review to evaluate completed work and reflect on the wins and losses throughout the project

**Kanban:**

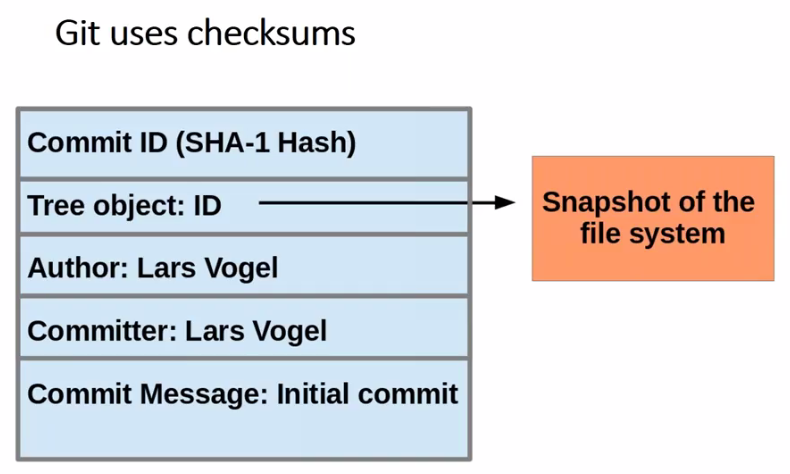
* Kanban is nothing to do with the release
* It is part of the Operations where works on the multiple releases but not bother about the release

**VCS Terminology:**

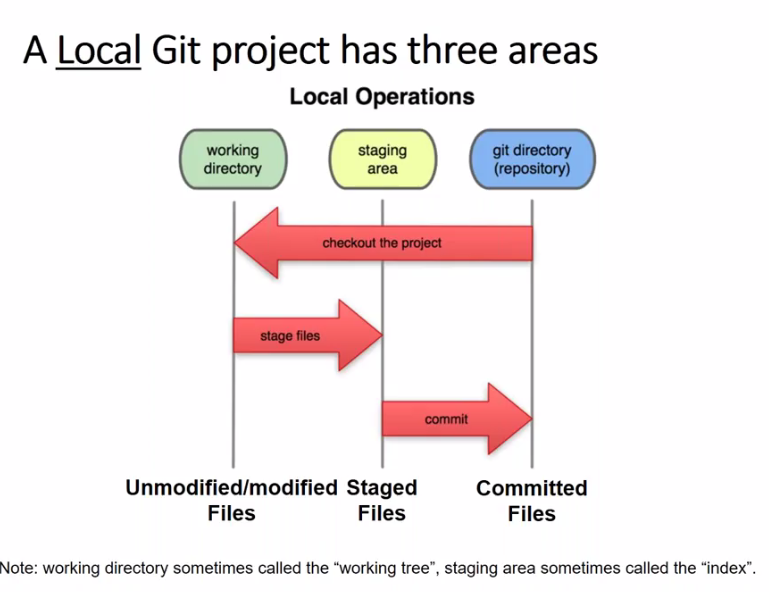
* Check Out: Taking a copy of the file from the repository to the folder of the local machine(Laptop/PC)
* Check In: Storing back the file into the repository from the folder of local machine.
* Folder on the local machine is called work space where the code or data will be modified or edited
* Every Check In will typically has an some information(meta data) which is called as versions

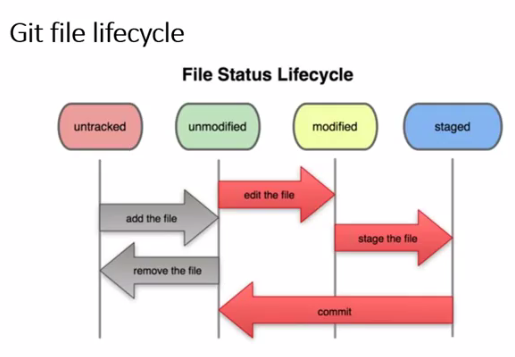


* Git takes a snap shots which consist of what was done earlier and what modification has done now in a file
* Git uses checksums



* Commit Id is a unique 40 character. It is a 64bit value.

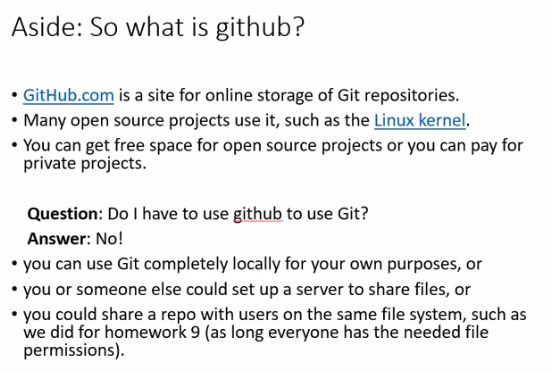




In Git there are two types of repositories

1. Local Repository: For storing on local machine and to do modifications to the file
2. Central / Remote Repository:: This is only for storing and sharing
3. Git is client side tool which is available for each development and users machine to create work space, checkout the code, modify the code, check-in and share the code.

Central repository manager tools are Github, Git lab, Git bucket etc.



Create an account in https://github.com/ using existing gmail account.

Path: <https://github.com/Elijah-Aleti/Learning.git>

Local repository commands:

Commands for git bash(Linux) terminal on local machine

$ which git ------> which command locates an executable file in your shell’s search path.

$ git version ------> shows the current installed version of git on local machine.

$ git config - -global user.name “Full Name” ------> to configure the username

$git config - -global user.email “email id” ------> to configure the email id

$ git config –global push.default “simple or upstream or current “ ------> to configure the name

$ git clone <source-centralrepo> <dest-localrepo>

Example: git clone <https://github.com/Elijah-Aleti/Learning.git> practice

where url is central repository and practice is local repository folder

If destination folder is not given then by default the folder name(Learning) in central

repository will be taken by default

$ git status ------> to show the status of repository like the commit status, untrack files etc.

$ git add <file name> ------> to save/load files on working directory into staging area of local repository and takes the snap shot.

$ git add < **.(dot) >** ------> to add/save/load all files on current working directory into staging area of local repository. Where dot represents current directory.

$ git commit –m “message” ------> to add files from staging area to repository

ls –a ------> to see all the files in the directory including hidden files/folders

* .git is local repository staging area.

$ git log ------> shows all the commit ids’ of the check-in’s of current/local repository

$ git log -1 ------> shows one commit id

$ git log -3 ------> shows latest one commit id; -3 shows latest two commit ids’

$ git log --oneline ------> shows commit id & message in short

$ git show <commit id> ------> shows all the details of the commits

$ git push ------> to update/share the changes done in local repository as commit id onto central repository

------> pushes only those commit ids’ which are not available on central repo but available on local repo.

$ git pull ------> to update/share the changes done in central repository as commit id onto local repository

------> pulls only those commit ids’ which are not available on local repo but available on central repo.

**Branching & Merging commands:**

$ git branch ------> shows the branch that now working

$ git branch <new branch name> ------> creates new branch

$ git checkout <branch name> ------> switches from one branch to other

$ git merge <source-branch> <dest-branch> ------> merges destination branch into source branch

# #Merge conflict

$ git cherry-pick <commit ID> ------> to do selective merge i.e., merging destination branch to source branch selectively or one by one file

**Stashing committed changes:**

$ git stash ------> creates stash, remove changes from working directory.

------> undo the changes and takes the backup of the changes in file before adding to the staging area and committing to the local repo

$ git stash list ------> to list all stash available for the repository

$ git stash pop ------> reapply the changes, remove stash

$ git stash apply stash@{Index num}------> apply a specific stash from repo

$ git stash clear ------> remove stash from repo

#head reference:

* The HEAD in Git is a file that references the current branch you are currently on.
* HEAD file stores the reference of the current branch.
* Every time in any repo, because of this head reference we can able to work in a

branch. It always points to the latest commit id.

$ git symbolic –ref HEAD ------> to read the value of he HEAD. This is alternate to $ git log command

**Git Reset:**

Git Reset is to undo the modified file which is in the working directory, staging area & head

reference in the repo before commit.

$ git reset –soft (repo reference) ------> this mode is to undo only from the repo reference i.e., from head reference to go back to the previous

reference.

$ git reset –mixed(repo + staging area)------> this mode is to undo from commit area and staging area

$ git reset –hard (repo + staging area + working directory) ------> this mode is to undo from commit area and staging area

$ git revert <commitId> ------> this will undo all the changes even after the commit of particular change and keep all the files contents similar to the previous commit

# This will not remove the commit id but creates the new commit id on top of the current

commit id and undo the changes in the file of particular current id.

$ git checkout <commitId> ------> this will get back to the changes committed before applying the command “git revert“.

# Here the commit id is the id which was generated after the changes done in the file.

**Tags:**

* Git has the option to tag a commit in the repo history so that you find it easier at a later point in time.
* Tag is a reference name/pattern applying to the particular commit id.
* The naming convention to follow is nothing but a pattern. Every organization has its own convention.

$ git tag –a <pattern> -m ‘comment’ <commitId> ------> Apply tag to a commit

$ git show <pattern> ------> Contents of the tag

$ git tag ------> displays the list of tags available

$ git push --tags ------> pushes the tags

$ git tag –d <tag> ------> deletes a tag

**Rebase:**

* Rebase is another kind of merge but it will not add new commit id but realigns branch.
* Step 1:
* Create a test branch
* Create a commit ID on master
* Create 2 commit ID’s on Test
* Rebase:

$ git checkout test

$ git rebase master

Step 2: Rebase

$ git checkout master

$ git rebase test

**Push/Pull changes other than Central Repo:**

$ git remote –v ------> List the remote reference

$ git remote add <ReferenceName> <NewRemoteURL> ------> add a new Remote reference

$ git checkout <branchname> ------> checkout the source repo branch to a non-working branch

$ git pull <ReferenceName> <Branch> ------> pull a change

$ git push <ReferenceName> <Branch> ------> push a change

$ git remote rm <ReferenceName> <NewRemoteURL> ------> removes a remote reference

Bare repo & Non-Bare repo:

* Bare repo is a central repo which is used only for storing and sharing
* Non-Bare repo is a local repo which is used for storing, sharing and modifications.

**Ignoring files & Viewing logs:**

**.**gitignore ------> this is for ignoring certain files and directories

------> git never ignores files which are already tracked, so changes in the

.gitignore file only affect new files.

------> commit the .gitignore to the Git repo

**File version:**

$ git log

$ git log - -oneline - -grep “workspace”

$ git log branchname

$ git log -1

$ git clean –n ------> is to ask whether to clean/remove the log files or untracked files in the repo while build

------> asks for whether to remove forcibly

$ git clean -f ------> is to clean/remove the log files or untracked files in the repo forcibly

**SVN vs. GIT:**

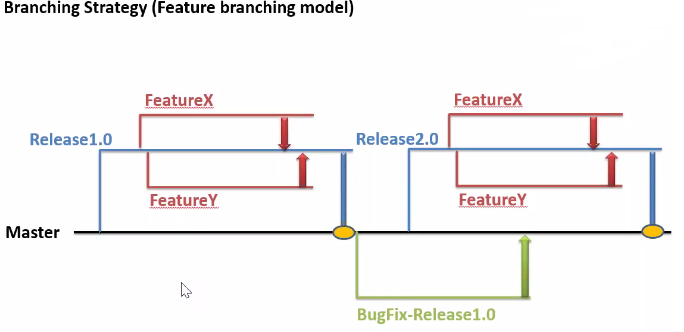
**SVN:**

* Central repo approach --- the main repo is the only true source, only the main repo has the complete file history
* Users check out local copies of the current version.

Git:

* Distributed repo approach --- every checkout of the repo is a full-fledged repo, complete with history
* Greater redundancy and speed
* Branching and merging repos is more heavily used as a result.

# Master branch is also called as Protected branch or stable branch,



* Develop: Program, Application, Script etc
* Unit Test/Smoke Test: Unit test is to test only one module or part of a program whereas smoke test is to test the entire develped application functionality whether it is working together as a product or application
* Review: Review the application or program
* Checkin: Push the code to central repo through the branch not through the master braanch or Product branch
* Review:

