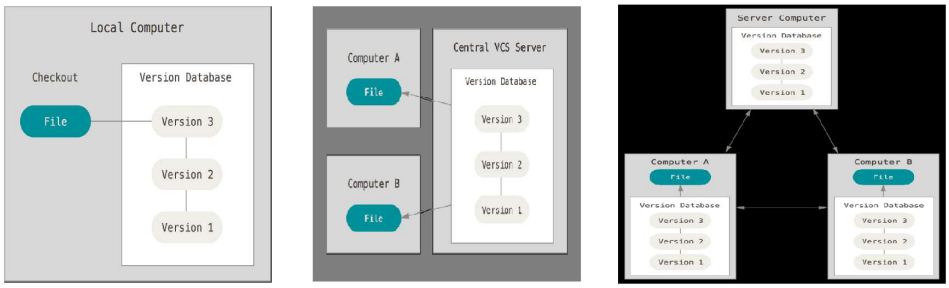
GIT

Thursday, May 12, 2022 8:45 PM

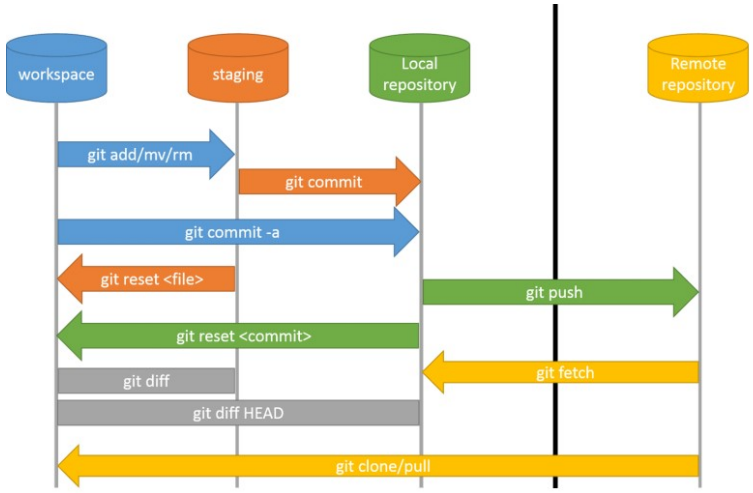
- **Version Control System (VCS)**
  - A VCS is a repository of files, every changes made to the files are tracked with revisions, along with who made the change, why they made it and references to problems fixed or enhancements introduced by the change.
  - It is also known as **Revision Control System (RCS) or Source Code Management(SCM).**
  - It help the team to ship the products faster, improves visibility, traceability for every change ever made, help teams collaborate around the world.

- **Types**
  - Local Version Control Systems.
    - Used in local computer for own purpose
  - Centralized version control systems.
    - used over the network. Repository is only at the server and every changes will be updated in the server thereby creates conflicts when multiple user do changes on the same file.
  - Distributed version control systems.
    - used over the network. Repository is at the server as well as each user has own copy in the local PC, so that once everything is done than we can push the changes into the server.



- **Fig : 1**

- **Git - Distributed Version Control System - Developed by Linus Torvalds**



- **Fig : 2**

- **Config**
  - **git config --global user.name "Arun"** ##### Global name and mail ID for multiple projects
  - **git config --global user.email "arun@gmail.com"** ##### Global name and mail ID for multiple projects
  - If we use different name for different projects than skip the **"--global"**

    Note : If you want to change the user name or email for certain project than go to that project and use below commands

  - **git config user.name "Arun"**
  - **git config user.email "arun@gmail.com"**

- **Pwd**
  - **Pwd** ##### Print current working directory

- **Touch**
  - **touch < filename >** ##### To create a new file

- **Init**
  - Make a git repository in local
    - **git init .** ##### To make the current directory as git repository (**"dot"** means current directory)
    - **git init < directory path >** ##### To make a specific directory as git repository

    Note : ".git" directory contains the meta data information of the git repository

- **mkdir**
  - **mkdir < dir name >** ##### To create a new directory

- **Add**

- To make the file available in the staging area which is nothing but just a memory. Here, after if we do some changes in that file than it will be tracked by version control system
  - **git add .** ##### **"Dot"** means all the files in the directory will go to staging area
  - **git add < file name >** ##### Only that file which is specified here will go to the staging area

- **Status**
  - **git status** ##### To know about the status of the git repository

- **Unstage**
  - **git rm --cached < file name >** ##### To remove the file from staging area

- **Commit**
  - **git commit -m "Files added"** ##### To add the files from staging area to local repository, "**-m**" is used to add message for the current commit

- **Log**
  - **git log** ##### Will give us the detailed information about the files in the local repository like who committed and when.
  - **git log --oneline** ##### Will give us details in shorter format

- **Restore**
  - **git restore < file name >** ##### Consider that we have a file already in the local repository and we did some changes to it. In **"git status"** we will come to know that changes are done to the file which is already in VCS. If we want the old file itself than we can restore or we can add the new changes to staging area than commit.
- **Cat**
  - **cat < file name >** ##### To view the file in the command prompt itself

- **Reset**
  - **git reset < commit ID >** ##### Reset the "**HEAD**" to different positions

    Note : Reset will remove the commit from the VCS like pop operation. **Ex:** Consider that we have 10 commits, if we reset to commit 2 than from commit 3 to commit 10 will be removed from VCS like pop.

  - **Soft Reset**
    - **git reset --soft < Commit ID > or git reset < Commit ID >** ##### Consider the new commit is 5 and old commit is 4, now reset to 4 will remove commit 5 data from VCS but data Will be available in the directory.
  - **Hard Reset**
    - **git reset --hard < Commit ID >** ##### This will remove the commit 5 from VCS as well as the data from the directory

  Note : By default **"--soft"** will execute if we use **git reset < Commit ID >**

- **Revert**
  - To delete a particular commit alone than we can go for **revert** option, in this we can track who deleted the commit. Using **"git log"** we can see the information about who deleted the commit but we will not be able to see the file changes or files added in that respective commit. **Ex :** Consider that we have 5 commits, if we want to delete the commit 3 alone than we can use this option. After the revert we will have 1, 2, 4, 5 commits only.
    - **git revert < commit ID >** ##### We need to give commit message after revert so that it can track who did the revert and why. Then, it will commit the changes automatically.
    - **git revert < commit ID > --no-commit** ##### Only remove the commit ID as well as the changes in that commit than we have to commit manually.

  Note : If we use **"git log"** than we can still see the commit ID which we deleted but only the contents will be removed.

- **Git - Remote Repository**
  - **GitHub / Bitbucket / GitLab - Public / Private Cloud Repositories**
    - Git uses some repository management services like GitHub, Gitlab, Bitbucket etc.…
    - These are repository hosting services, but it adds many of its own features. While **Git** is a command line tool, **GitHub** provides a Web-based graphical interface. It also provides access control and several collaboration features, such as wikis and basic task management tools for every project.

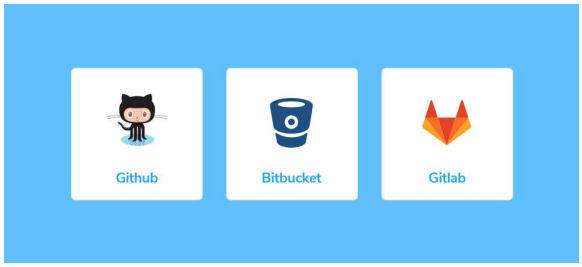


Fig : 3

- **Clone**
  - **git clone < url ID >** ##### To download the content of remote repository to local repository for the first time. **url** can be of **https**, or **ssh**, **ssh** is like we need to create a key in local repository and use the same key in remote repository to sync the data. Recommended is https.

  Note : If the repository which we are downloading is public than it won't ask user name and password, if it is private repository than user name and password is mandatory.
- **Push**
  - **git push origin master** ##### Pushing the local repository content to remote repository where **"origin"** is the source and **"master"** is remote branch
- **Pull**
  - **git pull -r** ##### Sync the remote repository contents to local repository. We don't have to use clone every time to download all the contents, just sync only the changed contents using pull

    Note : **" -r "** means remote

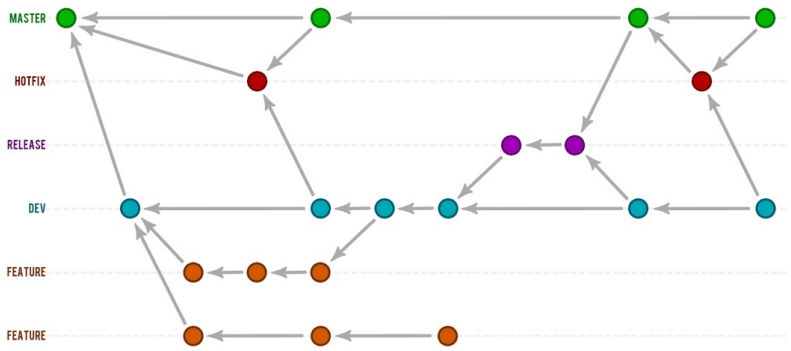
- **Remote**
  - **git remote -v** ##### To know which repository we are working on

    Note : It will show **fetch** and **pull** repository link. Sometimes we pull from one repository and push the contents to another repository that time we can find its useful.

- **Change the url**
  - **git remote set-url origin < url >** ##### To change the **url** for both the **fetch** and **push**
  - **git remote set-url --push origin < url >** ##### To change the **url** for **push**
  - **git remote set-url --fetch origin < url >** ##### To change the **url** for **fetch**

- **Add local repository to remote**
  - **git remote add origin < remote repository with the same as local repository name >** ##### We already have a local git repository but we don't have remote for it than we can create remote repository with the same name as local repository. Use these command to create a remote repository.
  - **git branch -M master** ##### Move to " Master branch ". We can check what branch we are in by using **"git branch"** command. If we are already in **"Master"** branch than we can skip this.
  - **git push -u origin master** ##### To push the local repository to master where **"-u"** is update.

- **Branching**



  - **Fig : 4**

- **Branch**
  - **git branch** ##### To know which branch we are in currently
  - **git branch -a** ##### To know which branch we are in currently, with some extra information
  - **git branch < branch name >** ##### To create a branch
  - **git checkout < new branch name >** ##### To move from the current branch to new branch

    Note : We can identify the current branch by using git branch command where the asterisk symbol points to current branch

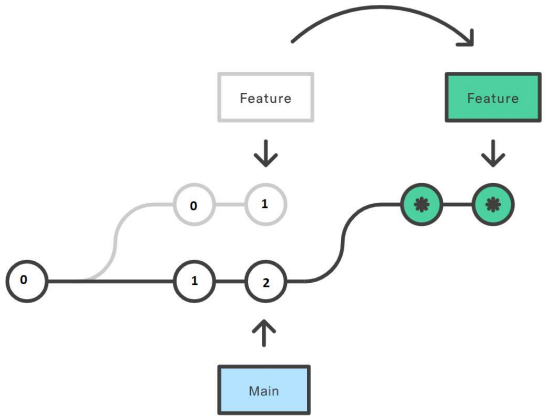
  - **git checkout -b < new branch name >** ##### To create a new branch and move to that branch
  - **git branch -d < current branch name >** ##### To delete the branch

    Note : Make sure we are in the current branch for deleting the current branch

- **Merging**
  - **git merge < old branch > < current branch >** ##### Merge the contents from the old branch to the current branch

    Note : Make sure we are in the current branch before executing the merge commands

- **Rebase**
  - Rebasing is the process of moving or combining a sequence of commits to a new base commit. Rebasing is often used as an alternative to merging. Rebasing a branch updates one branch with another by applying the commits of one branch on top of the commits of another branch.

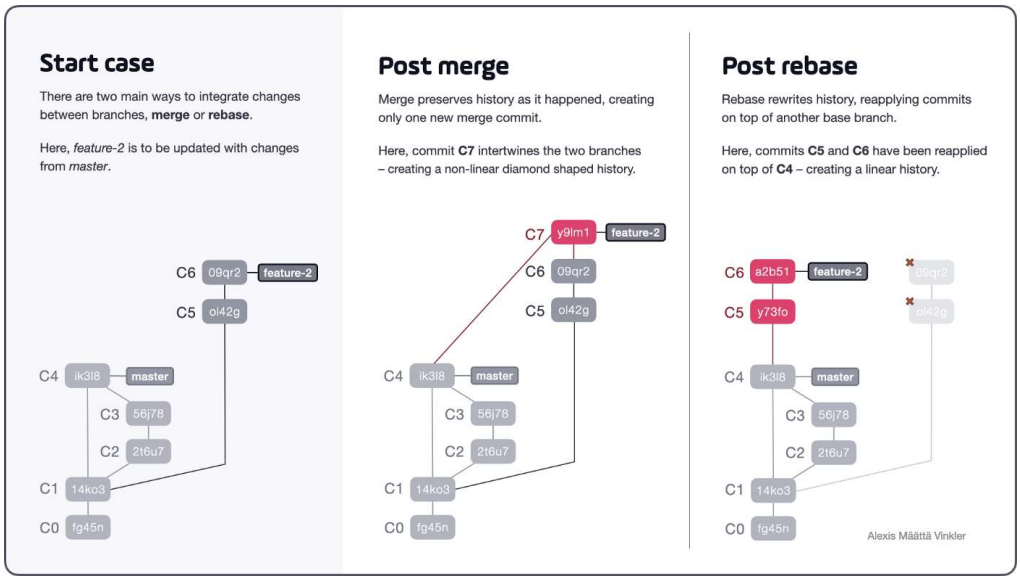


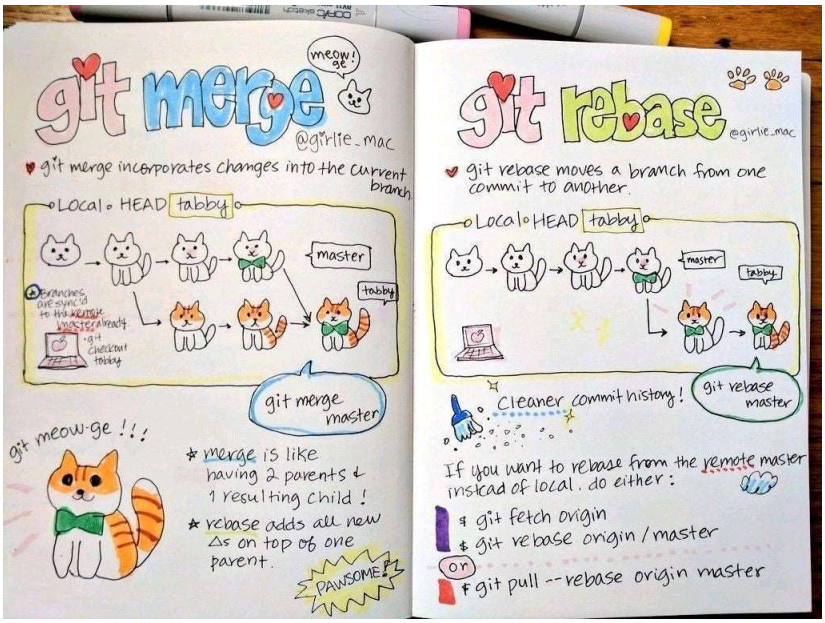
  - **Fig : 5**

  - Consider that we have created a feature branch from main "0" and did some commits like "1". Parallelly there where some changes going on in main as well which is 1 and 2. If we want the feature changes after the main than we have to use "**git rebase".** Where we will get the feature 1 changes after the main 2.

  - **git rebase < feature > < master >** ##### To rebase the master with the content of feature. Make sure we are currently in the master branch.

- **Difference Between Merge vs Rebase**





- **Cheryy-pick**
  - Cherry Pick is the act of picking a commit from a branch and applying it to another. **"git cherry-pick"** can be useful for undoing changes. For example, say a commit is accidentally made to the wrong branch. You can switch to the correct branch and cherry-pick the commit to where it should belong.
  - **git checkout master**
  - **git cherry-pick < commit ID >**

- **Stash**
  - Stashing is the process of storing our uncommitted changes into the some memory area, where it will show nothing in the git status command.
  - **Ex :** Consider a scenario where you are working on a **project A** in your working directory and you did some 1000+ changes as well as added lot of files. Now, suddenly you got some urgent task from a **project B** but you are in the middle of **project A** so you can't commit all the files. Now you can use stash and then move to the new **project B** branch finish that work, once it is done than you move back to **project A** to continue on it.
  - **git checkout project-A** ##### Move to the project A
  - **git add .** ##### Add the changed files and newly added files to staging area
  - **git stash** ##### Stash the files and contents to memory as we are moving to another branch
  - **git status** ##### We will not see anything as all changes are moved to memory
  - **git checkout project-B** ##### move to the project B
  - **git add .** ##### Add the changed files and newly added files to staging area
  - **git commit -m "project-B all changes are done"** ##### Commit the changes with the proper message
  - **git pull -r** ##### Sync the remote to local using "git pull"
  - **git push origin master** ##### Push the content of local project-B to remote
  - **git checkout project-A** ##### Move to project-A as the project-B is completed
  - **git stash list** ##### Check list of stashes which were done already
  - **git stash < stash ID >** ##### Get the specific stash from memory to current working directory
  - **git stash pop** ##### If we use this option than last stashed contents will be copied from memory to current working area
  - **git add .** ##### Add the changed files and newly added files to staging area
  - **git commit -m "project-A all changes are completed"** ##### Commit the changes with the proper message
  - **git pull -r** ##### Sync the remote to local using "git pull"
  - **git push origin master** ##### Push the content of local project-A to remote

Note : Stash IDs are from 0,1,2 …… ranges

- **Squash**
  - Squash is the process of combining multiple commit IDs into a single commit ID.
  - We don't have a specific command for squash, instead we use rebase interactive method.
  - **git rebase -i HEAD~[6]** ##### Squash the last 6 commits in the branch. After, this we will have only 1 commit ID instead of 6 Commit IDs.

    Note : **"-i"** means interactive, HEAD~[6] means from head 6 commits but we have to count the number commits which we did. In projects we might have done "n" number of commits which is quite tedious to count the number of commits. So we use still which commit we have to squash.

  - **git rebase -i < Commit ID >** ##### Squash till that commit ID

    Note : After Squash command we will get an interactive window where we have to replace the **"pick to squash"** except the commit ID which we entered for squash. Press Ctrl + O than Ctrl + X for save and exit. After this our commits will be converted into a single commit.