Git was developed by the person who created linux. Hence Git is designed similar to linux file system

* **git init** –creates .git folder
* **git status** – shows status of changed and untracked files
* **git add** – adds files to staging area
* **git commit** –m ‘message’
* **git log** – shows all the commits with commitID
* **open .git/objects** – opens folder where all the objects are created. Git objects can be blob, tree, commit, tag
* **git cat-file -p c34c6995ff8bd63a33780c6f7f1213930b928176** – shows the meta data about the committed changes. Since all the commits are linked when you do this command you will see the parent tag too that is the earlier commit before this one.
* **git tag tag1  -m 'stable version’ -** Tag is a label for a particular state. this will show you the tag ID. Now if you do 'cat-file’ (cat-file tag1 [this will give you sha1])on that tag you will see that its the same tag that you see when you do 'git log’ that points to head
* **git branch** – gives the list of all the branches. .git/refs – contains all the branches. If you manually create a file in that folder a new branch will be created. Branch is just a refrence to a commit.
* **git branch branch1**  - Create a new branch named branch1. Now if you do — **cat .git/refs/heads/master** and **cat .git/refs/heads/branch1** will give you the same result
* **cat .git/HEAD**  - The file at this path in the .git folder gives you the current branch or the HEAD is pointing to. HEAD is a reference to a branch
* **git checkout branch1** – switch branches. We are switching form ‘master' branch to the ‘branch1’ Refrer screen attached. Now if you do **'cat .git/HEAD**' you will see that the HEAD is pointing to the branch1. Hence checkout means move the HEAD and update the working area. Refer the screen shot when you do some commit(007f) on the branch1
* **git push -u origin –all** – to push the branch to remote
* **git merge  branch1** – Perform this after switching to master(git checkout master).  This merges the branch1 with Master. Merge is a commit, but just one difference is that it has 2 parents[e268, 007f]. Refer the screenshot attached. If this causes cauflict in some of the files then you have to manually resolve the conficts of the listed files using below command.
* **git merge —abort**– when the process does not work
* **vim filename** e.g. vim readme.txt – this will open the file on the terminal, will show the conflicting changes and you would have to merge it. Then save it using  esc + :wq.
* Now if you do git status you will see that the readme.txt file (conflicted files) are not staged. Hence you would have to manually add those files using **'git add readme.txt’**. Then do git commit. This will merge the branches. Now if you do ‘git log’ it will show the last commit as merge. Doing cat-file on that commitID it will have 2 parents shown.
* Now if you do git merge master , it will do fast-forward.

**Detached head**

* **Git checkout commitID** We can directly checkout a commit instead of branch. Here commitID is any one ID listed with git log. In this case the HEAD will be detached. There will be no current branch at all. You can verify this by doing cat .git/HEAD . Now if you make any commits those will not be on any branch. Please refer screen. Now if I do git checkout master then HEAD will point to master and my commits will be lost. To save the commits you can create a branch with git branch branch name , then merge that branch. Otherwise after some time your commits will be garbage collected.

**Rebase:**Its not merging the branches but it rearranges the branches, so that it looks like one single branch.

* **git checkout branch1** – now your HEAD is pointing to branch1
* **git rebase  master** – this will actually rebase the branch1. It will detach all the commits from branch1(blue ones in image) and place over the master . Screen attached. It makes copies of all the commits in the branch1 except parent (as parent chnages) and applies to master. The old commits since will be detached, those will be garbage collected later on. Rebase is an operation that creates new commits. Rebase history looks cleaner. Merge preserves the history.

Diff between git fetch and git pull. git pull = git fetch + git merge

**Git workflow:** Divided into 4 phases – STASH, WORKING AREA, INDEX, REPOSITORY

* **git diff** - shows file difference between working and index area
* **git diff –cached** – shows file diff between the index and repository