How Git Works

- **Git is a key value pairs at this low level**
  - **Value**
    - Any sequence of Bytes could be value.
      - Example: binary files, Sequence of characters, File ,etc..
  - **Key**
    - The key is a hash and they are unique
    - You give the Value for Git and it will give a hash (SHA1)
      - SHA1s are 20 bytes in hexadecimal format.
      - They are a 40 Hex digits
    - Keys will be used by Git to store this content in the map.
    - Can be Generated using:
      - echo "test" | git hash-object -stdin

- **Git is a Map**
  - **Git is a preistent Map**
    - **The reason for this since it can store the key value pairs on disk as shown:**
      - **Taking a look at the .git directory and specifically .git/objects (Object Database)**
        - **git init**
          - **In general, will create the .git hidden directory in the current directory**
          - **This will make the current repository a Git repository project**
        - Inside of the two digit directory there is a file which is the rest of the hash generated by previous command
          - File Name: f6c47d25426dee1f19922d555044a875ff f056f
          - The value created is a directory and inside that directory is a file
            - Because not all the stuff created will be inside one directory
            - **echo "test" | git hash-object -stdin -w**
              - the -w will right it to the current directory and give us the hash
              - Output: 2ff6c47d25426dee1f19922d555044a875ff f056f
        - the first 2 digits of the SHA1 generated are a directory
          - Directory Name:2f
        - **git cat-file 2ff6 -p**
          - **git cat-file 2ff6c47d25426dee1f19922d555044a875ff f056f -p**
            - Print the content of the file and will display "test"
        - **git cat-file 2ff6 -t**
          - **git cat-file 2ff6c47d25426dee1f19922d555044a875ff f056f -t**
            - Print the type and it will prting 'Blob'
    - **Git is a Content Tracker**
      - **When doing a git commit, we can see that it created a SHA1 for the commit**
        - **git commit contains:**
          - tree is a directory in Git
            - **tree and it's hash**
          - **Author and the date it was created**
          - **Message of the commit**
        - **git tree contains:**
          - **blob (meaning files) and it's hashes and permissions and names**
            - NOTE: blob with the same content will have the same hashes.
            - Meaning tree will point to the same blob if information is the same inside two different files.
          - **tree(meaning directories) and it's hashes and permissions and names**
            - **040000 tree 90d2dedf192e73e6f0f19b1ac4b2ed885b6cefde2e Recipe 100644 blob 2399189761e47e00da0d91a608f2c31d2fe0cbe5 menu.txt**
        - **git count-objects**
          - Will help us see how many objects exiss
          - **Each commit , tree or blob is an object in .git/ object directory**
      - **Three Rules of Git:**
        - **1- Current Branch track new commit**
          - If you do a new commit, the current branch moves to the new commit
          - If you are in detached HEAD, the the HEAD moves to the new commit
        - **2- When you move to another commit, Git updates your working directory**
        - **3- Any commit that doesn't have a branch, or directly connected head to it will be garbage collected.**
      - **How to merge two branches?**
        - **git merge lisa**
          - **Before**
            - HEAD master e268 007f lisa 5720 1177
          - **After**
            - HEAD master e268 007f lisa 5720 1177
            - **git cat-file -p ecbe**
              - **two master branch instead of one**
          - **Conflict**
            - We get a conflict and inside the conflicted file we try to solve the conflict
              - under it HEAD stuff
              - <<<<HEAD
              - under it lisa stuff
              - >>>>lisa
              - Those are separated by =======
            - **git add <conflicted file>**
              - added explicitly to say that the conflict is removed.
            - **git commit**
          - **No conflict**
            - Directly do everything no need
        - **Characteristic**
          - **1- Can become untraceable when large project is used.**
            - Example on Mac: store
            - on mongodb too much branches cannot know what is going on
          - **2- Preserve History of changes**
      - **How git rebase work?**
        - **git rebase master**
          - **Note: The HEAD is pointing at spaghetti since green**
          - **Before**
            - master spaghetti
        - **After**
          - **1- Git Search for the common parent branch which is the one in RED**
          - **2- Git Detaches the Whole spaghetti branch(Illusion)**
            - a- Create new commit for the blue commit pointing at the same Tree for each.
            - b- The Commit after the RED will have a parent the master branch.
            - c- The other commits in the spaghetti will have parents
          - **3- The Old commit are garbage collected**
            - x_branch
      - **What is Fast Forward?**
        - **git merge master**
          - **Note: Nothing has changed in Lisa when we did the merge for the fastforward to work**
          - **Before:**
            - master ecbe e268 007f lisa HEAD 5720 1177
          - **After**
            - **Fast Forward result**
            - master ecbe lisa HEAD e268 007f 5720 1177
        - **Characteristics**
          - **1- It helps refactor project history so it's nice to look at**
          - **2- It creates new commits and destroy old commits which will hinder the history of a project**
            - **It's a lie. It looks like the Yellow ones are created first and the blue are second. But that could be wrong.**
    - **Github.com stuff that are not git stuff**
      - **git fork**
        - Will create a copy of a remote repository to a repository that we own
          - Same as git pull but will be copied to our github.com account
        - **After:**
          - Same as pluralsight but on nusco account on github.com
          - Fork
            - pluralsight/nicepropject
            - nusco/nicepropject
      - **git clone nusco/ nicepropject**
        - **After:**
          - **git will point at the nusco/nicepropject and any changes on nusco/nicepropject can get it by git fetch on our own.**
          - **From git point of view there is no link between our own project and github.pluralsight.project at all**
          - **Github.com does know that these two project are connected but Git doesn't**
          - origin
          - nusco
      - **git push to our own project**
        - **After:**
          - origin
          - nusco
          - **Git will create a link between our project and the remote project**
      - **git pull from the upstream**
        - **After:**
          - origin
          - nusco
          - **we will have multiple remotes in which:**
      - **We add another remote to the project and we will call it upstream**
        - **After:**
          - origin
          - nusco
      - **git push to our remote**
        - **After:**
          - origin
          - nusco
          - **can sink between the forked and local**

- **Git is an Onion**
  - The reason for this because it's a layered like an onion
  - Layers
    - **1- Distributed Revision-Control System**
      - Not like any distributed control system do but does in a DISTRIBUTED way
    - **2- Revision Control System**
      - Dull a complex beast that includes: history, branches,merges
    - **3- Content Tracker**
      - Tracks content of files directories.
      - The definition of Git is this
    - **4- Map**
      - At it's core, Git is just map, a simple structure of Key Value Pairs
      - It's presistent and it is stored on disk

- **Git Objects**
  - **Blob**
    - Meaning a file
  - **Tree**
    - Meaning directory with permissions
  - **Commits**
    - Commit the project and submit the changes.
  - **Branches**
    - Branches is a reference to a commit
      - They move as we commit
    - Is like a branch for the current state of a project.
    - The only difference Tags don't move as the Branches does
  - **Tags**
    - Two Types
      - Regular Tags
        - **git tag <nameofthetag>**
      - Annotated Tags
        - Are the ones that comes with a message,who created,time and all
        - Can be created using:
          - **git tag -a <nameofthetag> -m "Message"**
    - **git tag**
      - To list the tags
    - **Created in the .git/refs/ tags**
      - **Cat .git/refs/tags/ <nameofthetag>**
        - OUTPUT:SHA1 object

- **Git Branches**
  - **a Branch is a reference to a commit**
  - **Master Branch**
    - **This Branch is created by Git to let us know where was our last commit**
    - **Created in .git/refs/ heads/master**
      - File contains the SHA1 of the commit and nothing else
      - Can be edited directly and we can use cat on vim on it directly
  - **git branch <nameofthebranch>**
    - To create a new branch
      - **This will create a file name of the branch in .git/refs/heads/ <nameofthebranch> and it's content would be the last commit**
  - **git branch**
    - To get the list of branches
      - (branch1)
      - **The asterisk means we are currently looking at this branch1**
      - **This is stored in the .git/ HEAD**
        - **ref: refs/heads/master**
  - **How Git commit work with branches?**
    - Before Commit
      - HEAD master e268 lisa 5720 1177
    - After Commit
      - HEAD master e268 lisa 5720 1177
        - **master in .git/refs/heads/<nameofthebranch> content change to new SHA1 commit**
        - HEAD still the same
  - **How to point HEAD to a specific Branch?**
    - **git checkout <name>**
      - Example: git checkout lisa
        - master e268 lisa HEAD 5720 1177
      - **Git will change the .git/ HEAD**
        - **ref: refs/heads/lisa**
  - **History and Contenx**
    - Graph
      - **Normal Commit**
        - **Blue:Blobs (Files)**
        - **Green:Trees(Directories)**
        - **Red:Commits**
      - **Merge Commit**

- **Detached HEAD**
  - **Before:**
    - HEAD master ecbe
    - **git checkout ecbe**
      - This will change the content of .git/HEAD to ecbe SHA1 hash
  - **After**
    - master ecbe HEAD
      - **git branch**
        - * (HEAD detached at ecbe0ef)
          - lisa
          - master
    - **git commit -m "Whatever"**
      - HEAD 7160 a8ef master ecbe
        - This will commit and points to the new commit
    - **git checkout master**
      - 7160 a8ef HEAD master ecbe
        - All will be lost if no branch is created for that last commit in detached head
        - warning: you are leaving 1 commit behind, not connected to any of your branches:
          - 7160d67 Remove sugar
          - diff3= red more option
          - If you want to keep them by creating a new branch, this may be a good time
          - to do so with:
          - git branch <new-branch-name> 7160d67
    - **git branch <nametebranch> 7160**
      - This will create a branch at the detached head before. The data won't be lost at that point.

- **Distributed Version Control**
  - **git clone <repositoryaddressorURL>**
    - **git/config**
      - repositoryformatversion = 0
      - (core)
        - filemode = true
        - bare = false
        - logallrefupdates = true
      - (remote "origin")
        - url =
          - https://github.com/ eneghavi/library
        - fetch = +refs/heads/ *:refs/remotes/origin/*
      - (branch "master")
        - remote = origin
        - merge = refs/heads/ master
    - **When issued git added some configuration into our repository in .git/ config**
      - When you clone a repository Git specify origin as default remote
      - You can add as many remote as you like
    - **git branch**
      - Will list the local branch and not all
    - **git branch -a#**
      - This will show all the references include the one on the remote
        - **This means all the branches listed on the .git/refs/remotes/ origin**
        - **If some are missing in the .git/refs/remotes/ origin Then they are in .git/packed-refs**
  - **git show-refs <nameofthebranch>**
    - Will show all the branches that have master in their names
    - this used to know if local and remote branch are similar
  - **git add .**
    - To add the stuff locally
  - **git commit -m "Message"**
    - To commit locally
  - **git push**
    - To push to remote
  - **What happens if there is conflict?**
    - **Option 1**
      - **git push -f**
        - Conflict!
          - origin master
          - local master origin/master
          - **Before:**
            - **Someone commited the remote**
            - **We commited locally**
          - **After:**
            - origin master
            - local master origin/master
            - **The remote matches the local**
            - **All the previous commits are deleted from the other person's commit**
            - **The green commit will be garbage collected if there is no tag on it**
    - **Option 2**
      - **git fetch**
        - **After:**
          - origin master
          - local master origin/master master
          - **Git will copy the origin or the remote branch and paste it into our local branch**
      - **git merge**
        - **After:**
          - origin master
          - local master origin/master master
          - **We solve the conflict locally and took into consideration the changes to the remote**
      - **git push**
        - **After:**
          - origin master
          - local master origin/master
          - **Git will push what we have locally to remote and preserve all history.**
    - **Option 3: (Same as two but one command)**
      - **git pull**
        - **git fetch -rgit merge**
        - **is as doing git fetch and git merge at the same time**