Git

* The First thing we need to do is set our username and the email
* Git has Multiple levels of Configurations

1. Repository /Project Level Configuration(Local)

Inside out git project repository .git folder is there inside that

folder config file is available repository/.git/config

1. User Account Level (Local)

In my system C:\Users\DELL inthis location it is there

1. System Level(Git installation)

In my case C:\Program Files\Git\etc where git is installed

To Check complete Git Config

git config --list –show -origin

Adding Project Level Username and Emial

* This is required because if we want use separate credentials for this particular project
* First we need to navigate inside the project folder and git bash there

git config --local user.email [kullaraki@gmail.com](mailto:kullaraki@gmail.com)

git config --local user.name rakesh

* If we want to remove this project level user and use system level user only

git config --unset user.email

git config --unset user.name

* To completely remove user section at time

git config --local –remove-section user

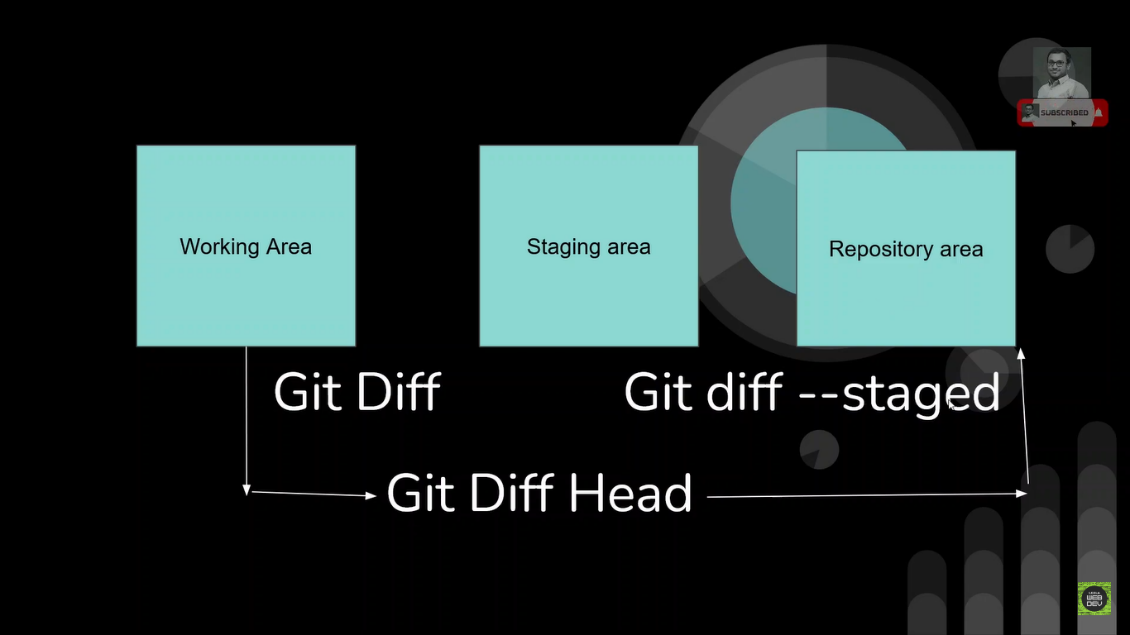
* To Untrack the Staged File

git rm --cached ManagerApplication.java

* Using Git Log Command

git log is used to check our previous commit and History

Git Diff Command



* To Check the Changes in the Working Area and the Staged Area

git diff

* To Chaeck the Changes in bitween the stage area and the repository area

git diff --staged

* To Chaeck the Changes in bitween the Working Area and the repository area

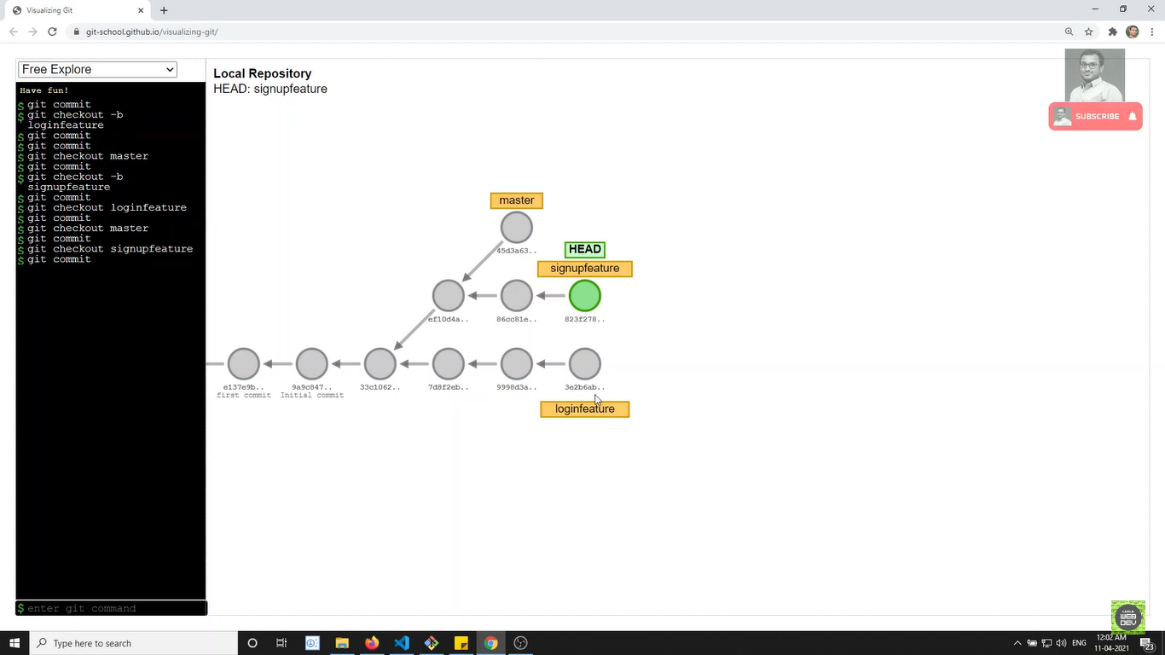
git diff head

How Data Stored In Git

* In Git Every thing is Stored in key and Values the key is SHA -1 and value is our data
* The data is stored inside .git -->objects folder
* To View the data in that file we need to use below commands

git cat-file c17174a9bb298d2b8dba098b6dc62fce8e41c141 -p

Git Branching



* To Create a New branch in Git we need to use

git checkout -b branchname

* Now Head is point to the newly Created Branch we can now commit the code to newly created branch
* If we want to come back to master branh we need to use

git checkout master

* If we want to rename a Git branch we need to use the below command

git branch -m signup\_feature

* If we want to delete a git branch we need follow the steps

1. First we need to move to any other branch because we cannot delete the same branch whre we exist

git branch -d signup\_feature

error: The branch 'signup\_feature' is not fully merged.

If you are sure you want to delete it, run 'git branch -D signup\_feature'.

1. If we alredy have some commits to that branch and not merged to master we need to use this command

git branch -D signup\_feature

* In Git all Branches are stored in this location

.git\refs\heads

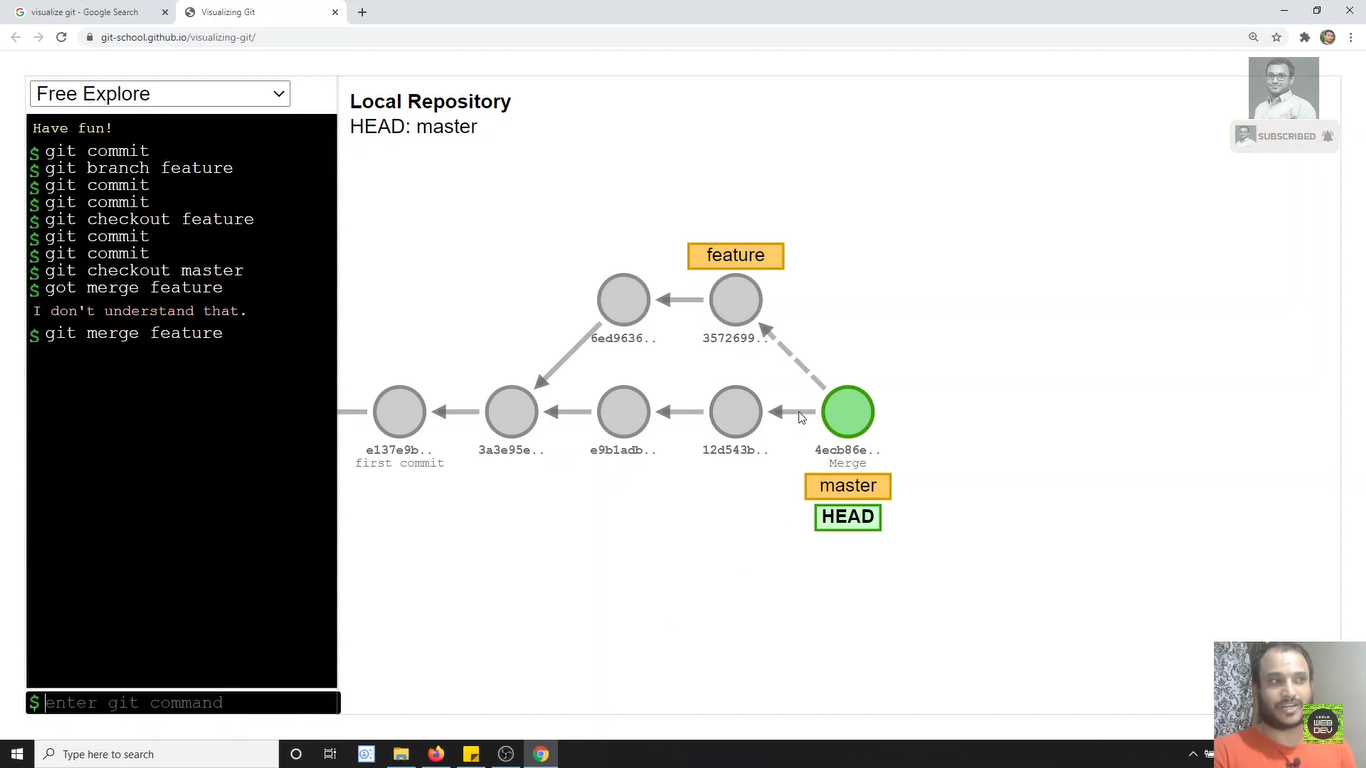
Git Merging

* To Merge new branch to master branch first we need to checkout to master branch then we need to use below command

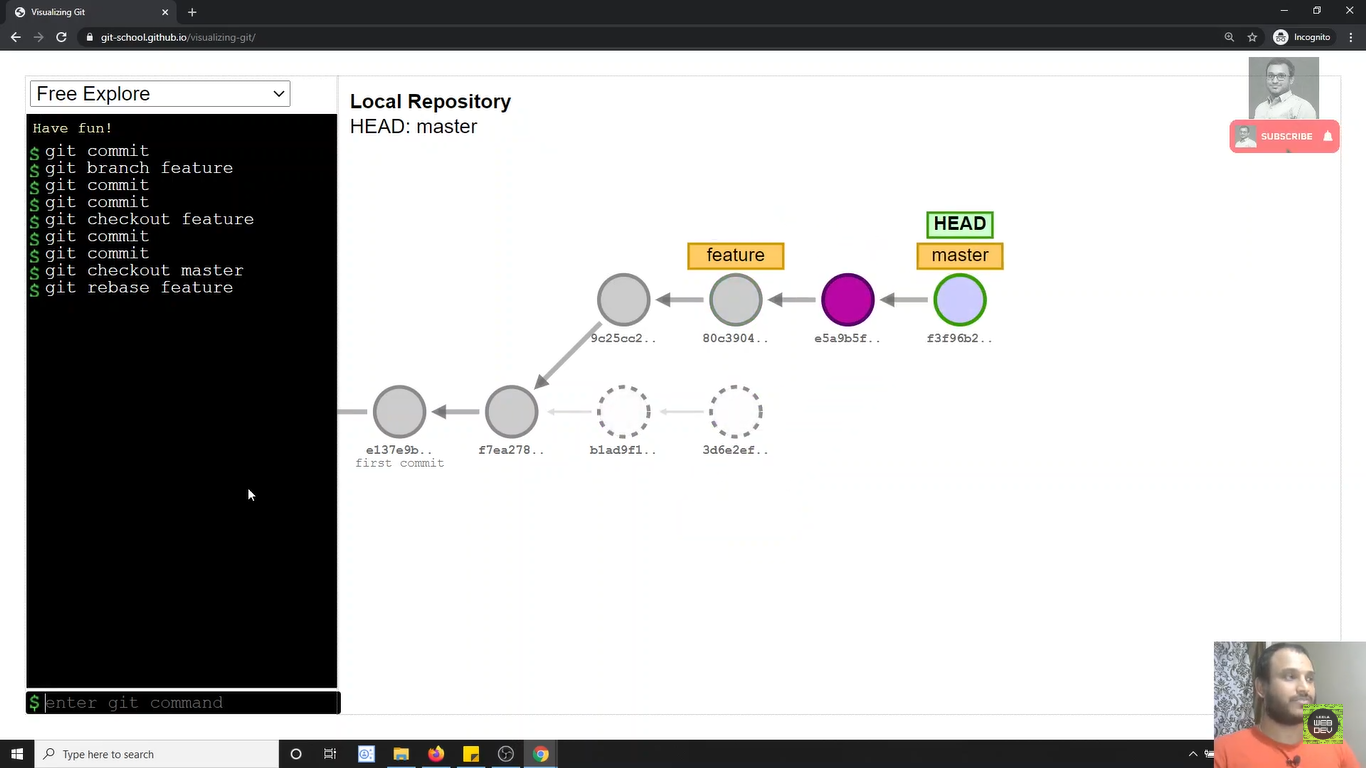
git merge new\_feature

Git Rebase

* Git Rebase is one of the Most powerful Git command
* Reabase is used as an alternative to merge
* Rebasing a branch updates one branch with onether with applying the commits of one branch on top of the commits of onter branch
* Git Rebase is used to Clean up our local commit history
* Rebase is advanced command which used rarly
* Merge preserve history
* Rebase dose’nt preserve history
* Don’t use Rebase when the branch is public when it is shared to all the developers Most of the teams prefers merge over Rebase
* Common places where we use rebase Cleaning up your commits before sharing your branch and pulling changes from onether branch with out merge



The above is example for merging



The above one is Example for Rebase

git rebase new\_feature

Modifieng Last Commit

* This command is Used only when the changes are not pushed to cloud or other developer
* This is used to modify recent commit if we mistakenly forgot something in that commit
* So instead of making a new commit we can append our changes to the last commit

git commit -m"edited command" –amend

Cherry-pick Command

* Cherry pick is used if you want to apply a perticular commit from one branch into other branch
* Cherry-pick is mainly used if you don’t want to merge the whole branch and you want some of the commits
* Cherry-pick just like rebase it is and advanced concept and also a powerful command
* Cherry-pick is a useful tool, but always its not a good option
* It can cause duplicate commits
* Mainly cherry-pick is used for bug fixes where you want to place that bug fix commit in all the version branches
* It also used when accedentally made a commit in wrong branch
* This works for example if we have three branches we made some changes in one branch and commited, if we want replicate that changes in remaining branches we need to checkout for that particular branch
* And we need to copy branch hash code and use below command

git cherry-pick 0a7e728

for reference

241 git branch

242 git branch 1.0

243 git branch 2.0

244 git branch 3.0

245 git checkout 3.0

246 git status

247 git add .

248 git status

249 git commit -m"new feature is added in 3.0"

250 git checkout 2.0

251 git log

252 git log oneline

253 git log --oneline

254 git checkout 3.0

255 git log --oneline

256 git checkout 2.0

257 git cherry-pic 0a7e728

258 git cherry-pick 0a7e728

259 git history

Git Reset Command

* The term reset itself stands for undoing changes
* The reset is often reffered as confusing command.
* Reset does different things in different contexts
* So That’s why its considered as complex command
* We Know that if we want to move the branch we use

Following commands

1)Commit

2)Merge

3)Rebase

* If you Observe Carefully These commands are not used

for moving the branch

* These are Moved as a side effet of creating new Commit
* Do we Have Any Command That Specially Used for Moving a

Branch

* Reset Does this. It is Used to Moved the branch
* Reset Has Options

--hard 🡪 Moves the files both to Working area and

Staging area

--mixed 🡪 Moves the files only to stage area.(default

Option)

--soft 🡪 does not move any files

We can Say that

Reset moves the current branch and optionally copies the

data from the Repositories to the working or staging area

git reset 955093a

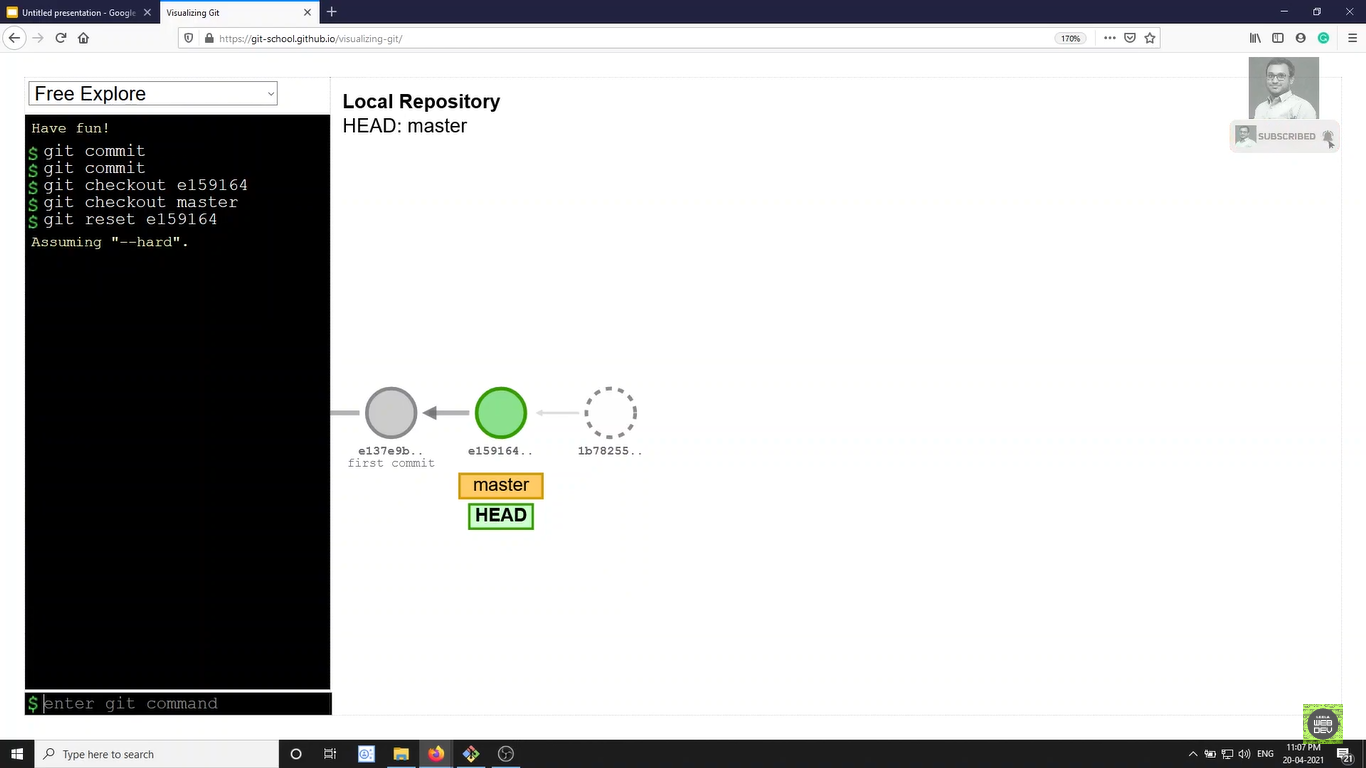
This command only moves the branch to given commit and brings

The files that are in repository to working area

git reset 955093a –hard

This command is used to moves the branch to the given commit

and removes the files completely from working area



Example of Git Reset

Git Stash

* Some times you want to Switch the branches,but you are

working on an incomplete part of your current project

* You Don’t want make a Commit of half-done work.Git

stashing allows you to do so.

* Normarlly when you switch branch you will commit the

Code and switch to the new branch

* If you switch branch with out commiting Two things will

Happened.

1)Switches to the branch carrieng the changes

2)Git will not allow to switch the branch and asks to

Commit or stash the Changes.

The Git stash Command enables you to switch branches

Without commiting the current branch.

The stash’s meaning is store something safely in a

hidden place. The Sence is Git is also the same for

stash; Git Temperarily saves your data safely without

commiting

If we create a new File and we want it to Stash First we

need to add for Staging.

git stash

If we want to add a name for stash then we need to use

git stash push -m"test stash"

If we Want to retrive Stash we can use two commands

1. Git pop

git stash pop

this command is retrive the changes and remove the

stash from the list

1. Git apply

git stash apply

this command is used to retrive the changes but not

remove the stash from the list,so that we can use the

stash again

git stash apply --index 0

this command is used to retrive the changes based on

index

Git Checkout

Git Checkout does lot of things

1. Moves from one branch to other

git checkout main

1. Creates new branch if does not existed and moves

head to that branch

git checkout -b newfeature

1. Also not only branch it also shits to particular

Commit hash

git checkout d37c72e

if we use the above command we are getting the below

message

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

So if we want to went back to previous state we can use

git checkout –

or we can use the below cammand

git checkout master

* Git Checkout also Supports another Syntax

git checkout head~2

Maens Without knowing Hash we can move the commit two step

back, if want to move back to previous state we can use

git checkout master

* Git Checkout also Used to discard changes in the file

You can revert the changes of the perticular file using

git checkout head filename

Or

git checkout –filename

* Git reset – hard can also used to rivert the changes but not selected files all files at a time

Git Switch Command

* Git Switch command is used move from one branc to onether branch and creates new branch

git switch 1.0

git switch -c rakeshbranch

* With the help of switch we cannot move head to particular commit hash

Git Restore command

* Git restore command is used to rivert the changes
* We can use git checkout also to rivert the changes but if we use this both files present in staging area and working area are riverted back
* So that’s why we are using git restore command to rivert changes in working area and staging area seperately

git restore manager/src/main/java/com/smart/controller/HomeController.java

this command to rivert changes in working area

git restore --staged manager/src/main/java/com/smart/controller/HomeController.java

this command is used to rivert changes in staging area

* Git Restore is also used to get the changes of particular

Commit we can use it

git restore --source Head~1 manager/src/main/java/com/smart/controller/HomeController.java

if we use the above command we can get the changes of HomeController.java from one commit before

Git Hub

Git hub with ssh

* We can Connect to git hub using two methods

1. Using Https
2. Using SSH

When working with a Git hub repository, you’ll often need to identify yourself to Github using your username and password

An SSH key is an alternative way to identify yourself that does’nt require you to enter your username and password every time.

To implement SSH key in the system

We need to create the SSH key pair using

ssh-keygen -t ed25519 -C "your\_email@example.com

we need to run the above command in git bash

C:\Users\DELL\.ssh

By defaut ssh public and private key generate in above

Folder

eval "$(ssh-agent -s)"

use above commad to check wether it is running or not

ssh-add ~/.ssh/id\_ed25519

Use the above commad in git bash to add ssh to agent

Then we need to go to git hub profile and click on settings and then

click on SSh and GPG keys

Then click on new key and give some title and paste our public ssh

Key below tab and click on add ssh key

Key will be added then in git bash

git clone git@github.com:ravennela/smart\_contact\_manager.git

Adding Remote

Git remote add origin <Url>

Origin is the short name for the url

* This means when ever I use the name origin, iam reffering to the particular github url like an alias name
* The name origin is the conventional git remote name, it is not at all special. Its just a name for a url.
* When we clone a Git hub repo, the default remote name setup for us is called origin. You can change it but most people leave it.
* How we have master as the default branch just like that orogin is default, if we needed we can change the name

We can also use like this

Git remote add mygithuburl <url>

* This means any time you use the name mygithuburl we are reffering to this particular git hub repo url

Renaming Remote Repo’s

Git rename <oldname> <newname>

We can also remove the remote using

git remote remove <name>

Pushing Code to Remote Url

* Not only To master we can also push the changes in the master or any branch to the different branch in git hub

Git push <remote> <localbranch>:<remotebranch>

Example

git push origin rakeshbranch:1.0

in this example commit in rakesh branch pushed to 1.0 branch

To https://github.com/ravennela/smart\_contact\_manager.git

\* [new branch] rakeshbranch -> 1.0

-u Option

* The -u option allows us to set the upstream of the branch

we are pushing.

* You can think of this as a link bitween the local branch

in git hub.

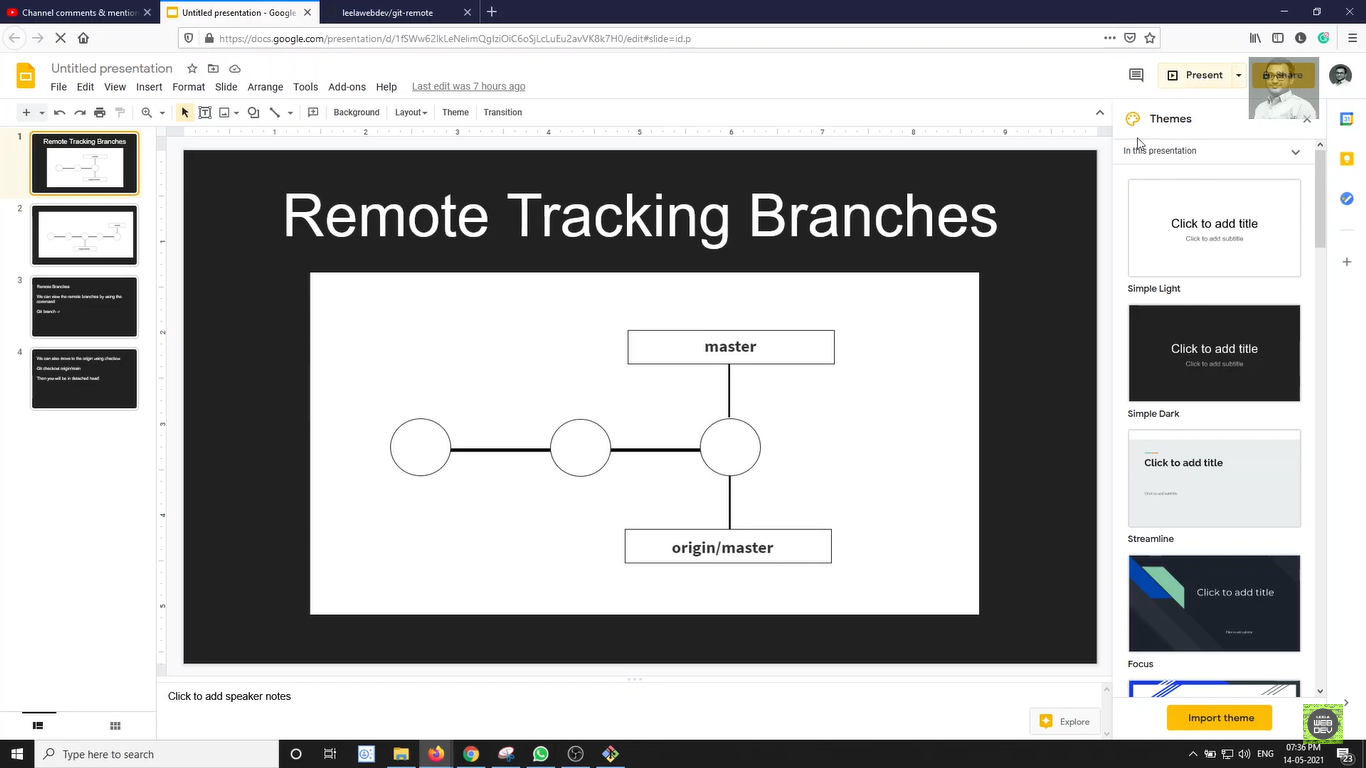
Git push origin master

* Running This command sets the Upstream of the local master branch so that it tracks the master branch on the origin repo.

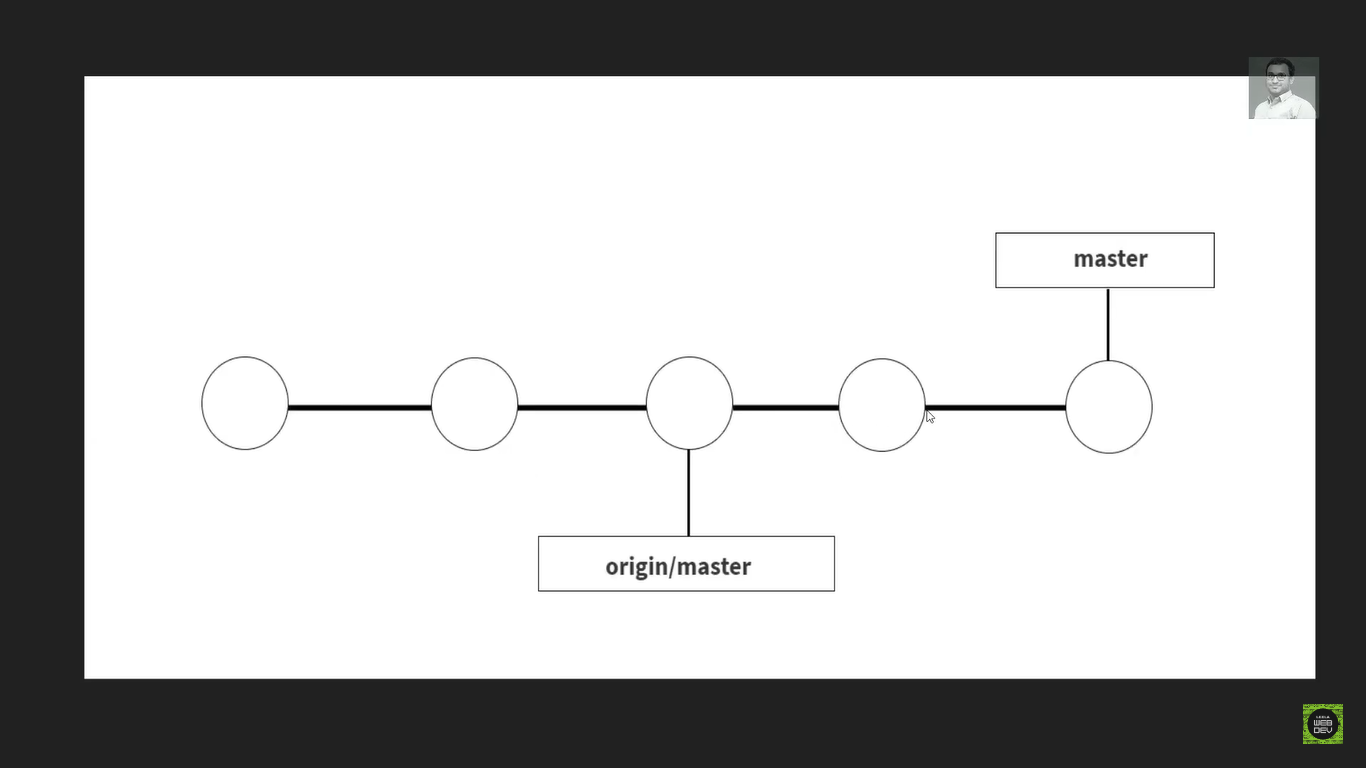
What is the use of this Option Then?

* Once we set the upstream for a branch, we can use the git push shorthand which will push our current branch to the upstream.

Remote Tracking Branches



* In the Above example The master in local repository and remote repository are pointing to the same commit that means the local and remote are in sync



* In the above example ofter make two commits the local master branch is two steps ahead of remote branch that mean we have two commits in our local we need to them to remote branch.
* When we clone a repository from github even we have multiple branches in git initially we are only connected to main branch
* To see how many branches present remote repository we need to use below command

git branch -r

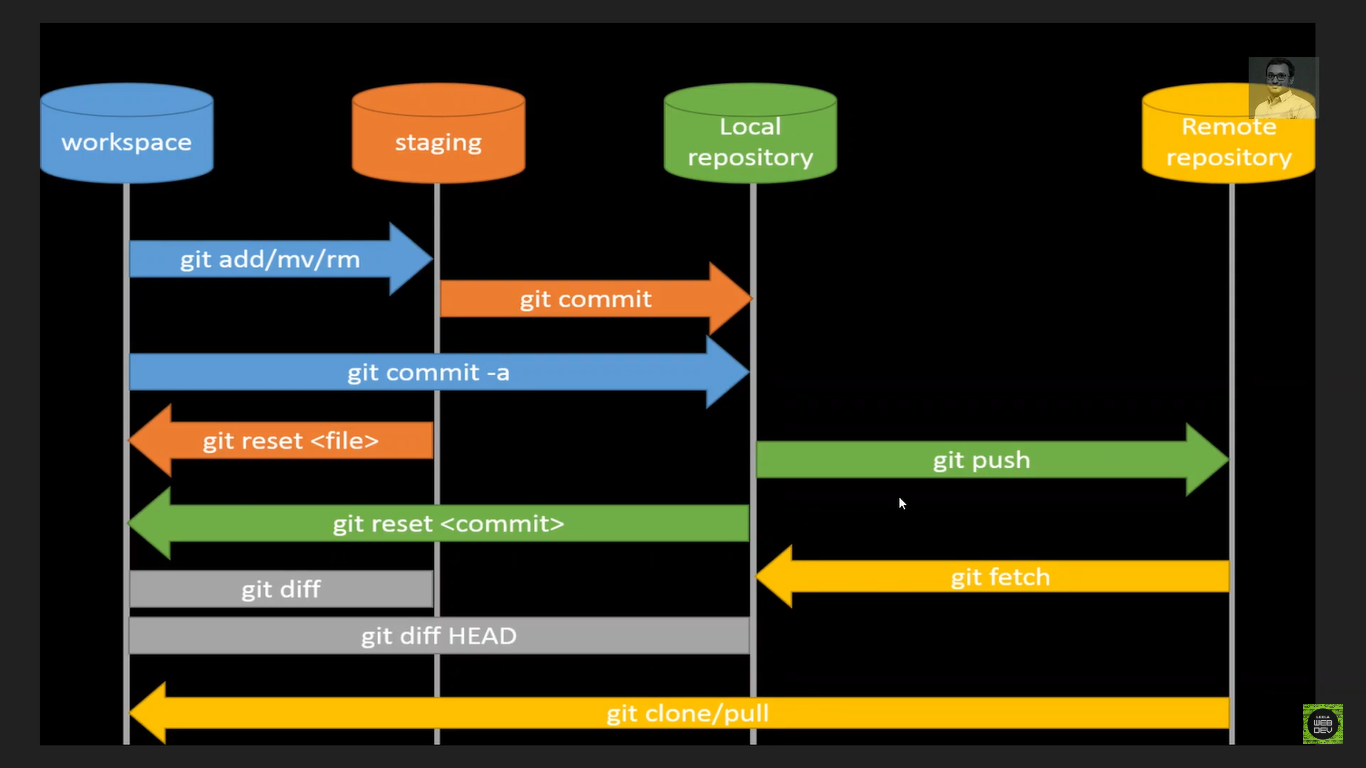
* If we want to change the branch from master to other branch we can use below command

git switch branch3

* this will switch the branch and make connection with remote branch

Git Fetch Command

* When you are working on with other collaborators on a github repo.
* One of your teammate has pushed up the changes to the master branch, but my local repo doesn’t know
* Then how do I get those changes???
* This is where git fetch and git pull come into play
* Git fetch and pull get those changes from git hub repo to your local repo .
* But these two commands has some differences.



* Fetching allows us to download changes from remote repository
* But those changes will not be automatically integrated to our working files.
* it just lets you see what others have been working in, with out merging those changes into your local repo.
* Just think of git fetch as link
* Please go and get the latest information from git hub but don’t add it into my working directory.
* Command for git fetch is

Git fetch <remote>

* This command fetches branches and history from a specific remote repository. It only updates the remote traking branches.
* Git fetch origin would fetch all changes from the origin remote repository.
* If not specified remote it defaults to origin.
* We can also Fetch a specific branch from a remote using the command

Git fetch <Remote> <branch>

* For Example **git fetch origin master** would retrive the latest information from the master branch on the origin remote repository.
* After fetching the changes. I will have those changes on my machine,
* But if I want to see those changes then I have to do the checkout to origin/master
* Your local master branch will be untouched.

Git Pull

* Pull Command is somewhat similor to fetch.
* Git pull is the command that we can use to retrive

changes from the remote repository.

* Unlike Fetch,pull actually updates our Head branch with

Whatever changes are retrived from the remote.

* In simple words we can say that
* Go and download data from github and immediately update

My local repo with those changes

We can say that

Gitpull=Gitfetch+Gitmerge

* Git fetch -Update the remote tracking branch with the

Latest changes from the remote repository

* Git merge -upadte my current branch with whateverchanges

are on the remote tracking branch

* Syntax for git pull

Git pull<remote> <branch>

* To pull,we specify the particular remote and branch we

want to pull using the git pull <remote> <branch> .

* What really matter is the where we are on. Whatever the

branch iam in that is where the changes will be merged

where iam pulling down to.

* Git pull origin master would fetch the latest information

from the origin master and merge those changes in our current branch

* Pull can results the merge conflicts.
* We need to resolve the conflicts just like normal merge.
* Some times you might have some work locally that is not on git hub and git hub have some commits.
* When you pull down, there may be conflicts.
* We also have shorthand syntax for the pull command like fetch

**>Git pull**

* If we run git pull with out specifying a particular remote or branch to pull from, Git assumes the follwing

1. Remote will be default to origin
2. Branch will defualt to whatever tracking connection is configured for the current branch.

Github Gists

* Github Gists are the simple way to share the code snippets and useful fragments to others.
* Gists are much easier to create, but offer few features compared to to normal Git Repository.
* Every gist is a Git repository, means it can be cloned.
* If you are singed into github when you create a gist that gist will be associated with your account and you will see it in your list of gists when you navigate to your gist home page.
* <https://gist.github.com/>
* Gists can be public or secret
* Public gists show up in Discover, where people can browse new gists as they are created. They are also searchable, so you can use them if you’d like other people to find and see your work.
* Secret gist don’t show up in discover and are not searchable.Secret gists are’nt private.
* If you send the Url of a secret gists to a friend, they can be see it.
* However, if someone you don’t know discovers the URL, they’ll also be able to see your gist.

Modify Branch Name

git branch -M main

Github Pages

* Github pages are the public web pages that are hosted and published by Github
* They allow you to Create a Website by Simply pushing your code to Github.
* Git hub Pages is a hosting service for serving static web pages
* It doesn’t support server side code like PHP, python, Ruby or Node.
* Github Pages supports only Html,CSS and Js Code only
* You get unlimited Projects sites in the github
* Each github repo can Have a corresponding hosted Website.
* It is as simple as just telling the github which branch to take for hosting the repo.
* The defualt url in git fallows this pattern

<http://username.github.com/repo-name>

* Remember you can get only one website for account or orgonisation
* But unlimited repo project sites.

Pull Requests

* Pull Request are a feature built in to products like Github & Bitbucket.
* They are not native to Git itself
* They allow developers to alert team members to work that needs to be reviewed
* They provide a mechanism to approve or reject the work on a given branch.
* They also help facilitate discussion and feedback on the specified commits.
* Pull Request is nothing but Merging in Feature Branches.
* At some point the work we did on the feature branch need to be merge into the master branch!
* There are couple of options for how to do this

1. Merge at will, Without any sort of discussion with teammates. Just do it what ever you want.
2. Send an email or chat message or something to your team to discuss if the changes should be merged in
3. Pull Request!

Pull Request workflow

1) Do some changes in locally on a feature branch

2) Push up the feature branch to Github.

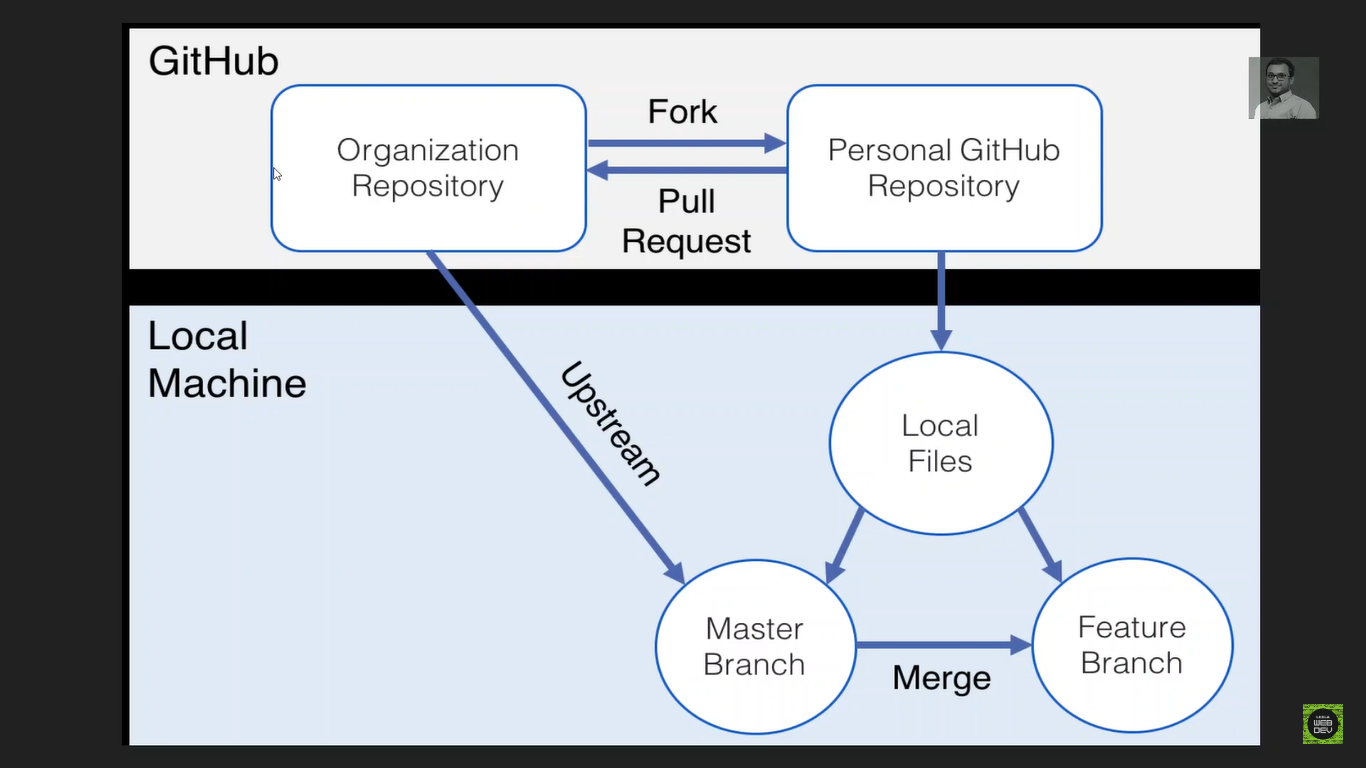
3) Open a Pull Request using the feature branch just

Pushed up to Github.

1. Wait for the Pull request to be approved and merged.Start a discussion on the Pull request.

Git Forking

* When you are having large open source projects with lots of contributers.
* They employ this forking strategy or workflow where there might be a handful of actual maintainers.
* They cannot add thousands of people as direct contributers or collabarators.
* This forking strategy enables anybody to try and make a contribution for the repository.
* There is no permissions needed. You can make your own copy. You try making changes and then you make a pull request.
* Any body can make pull request.
* As like pull request, forking is not a git native feature.
* The ability to fork is implented by github.
* I can clone my fork and make changes, add feature and break things without fear of distrubing the original repository.
* If I want to share my work,I can make a pull request from my fork to the original repo.
* This means that whole bunch of people can work on the project without actually having the permission to them.



* The fork and clone workflow might seem complicated, but its extremely common for good reason.
* It allows the project manager to accept contributions from developers all around the world with out having adding them as actual owners of the main project repository or worry about giving them permissions to push the repo.

Git Tags

* Git Tags main idea is that we can tag particuar commit, so we can lable commits by creating a tag, reference to a moment in time
* Tags are pointers that refers to a particular points in git history. We can mark a particular moment in time with a tag.
* Tags are most often used to mark version releases in projects (v4.0.1, v2.3.2, etc).
* Think of tags as branch references that to not change.Once a tag is created,it always refers to the same commit.
* Its just a label for a commit.
* There are two types of tags

1. Light weight tags
2. Annotated Tags

* Lightweight tags are light weight.They are just a name/label that ponts to a particular commits.
* Annotated tags store extra meta data including the auther’s name and email, the date and a tagging messages (like a commit message).
* Annotated tags however are stored as full objects in the git database.
* Its generally recommeded that you create annotated tags so you have full information.
* Light weight tag is much like a branch that doesn’t change -Its just a pointer to a commit.

Semantic Versioning

* Semantic Versioning specs outlines a standerdized versioning system for software releases.
* It provides a consistent way for developers to give meaning to their software release.
* Version consists of three numbers seperated by peroids. (4.2.1)



Typically the initial release(first release ) for any project will be 1.0.0

**Patch Realse (1.0.1)**

Patch realses normally do not contain new features or significant changes. They typically signify bug fixes and other changes that do not impact how the code is used.

**Miner Realse (1.1.0)**

Minor releases signify that new features or functionality have been added, but the project is still backwards compatable. No braking changes. The new functionality is optional and should not force users to rewrite their own code.

**Major Realse (2.0.0)**

Major Release signify significant changes that is no longer backwards compatable. Features may be removed or changed substantially.

Viewing Tags

> git tag

Git tag will print a list of all tags in the current repository.

* We can also search for tags that match particular pattern by using git tag -l and then passing wildcard. Forexample git tag -l “ \*beta\* ” will print a list of tags that includes beta in their name.

Git checkout <tag>

* To view the state of a repo at a particular tag, we can use git checkout <tag>. This puts us in detached Head.

Ex: git tag

git tag -l beta //to serach tag with name beta

git tag -l \*beta\* // to search all the tags that contains beta name

git checkout v17.0.9 //to checkout particular version tag

git diff v17.0.0 v17.0.2 //to find diff b/w two tags

Creating Light weight Tags

git tag v1.0.0

git show v1.0.0 // To display the details of the tag

Creating Annotated Tags

git tag -a v1.1.0 -m"annotated tag created"

Pushing Tags to Git hub

git push origin v1.0.0 //pushing specific tag

git push origin --tags //pushing all tags at a time

Reflogs

* The term Reflogs is a shortform for Reference Logs.
* These are just logs that git keeps us for as a record.
* Git keeps a Record of when the tip of branches and other references were updated in the repo.
* We can view and update these reflogs using the git reflog command
* Git only keeps reflogs on your local activity. They are not shared with collaborators
* Reflogs also expire. Git cleans out old entires after around 90 days, though this can be configured.
* The git reflog command accepts subcommands like show, expires,delete and exist
* Show is the only command used varient, and it is the defaults
* **git reflog show** command will show the log of a specific reference (it defaults to Head)
* For Example, to view the logs for the tip of the main branch we could run

git relog show main

git reflog show head@{2} //used to show logs from second commit

git reflog show main@{2} // used to show logs from second commit in main branch

git reflog show [main@{1.day.ago}](mailto:main@%7b1.day.ago%7d) //used to show logs one day before

* Reflogs are used to show the logs even if the commit is lost for example if use git reset ad79a8e –-hard then head is pointing to that commit, even we use git log --oneline that commit hash is not displayed.
* So with the help of reflog only we can get the particular hash and we can use git reset ad79a8e –-hard to again move the head to that required commit.

Git Aliases

* Git aliases is the short form for the git command in simple words we can say that nik name for person.
* To do this first we need to go to C:\Users\DELL location
* Then we need to open .gitconfig file.
* Then we need to add [alias] and below this line we can write our short hand code.

[alias]

s=status

lg=log –-oneline

* We can also do this with a command

git config --global alias.br "branch"

Git Revert Command

* Git Revert command is used to revert back the changes if we mistakenly commited the changes.
* Git reset also do the same thing but the main difference is if use reset the commit completely deleted

git reset ad79a8e --hard

* we are loose all the commits before that commit even we don’t have history also.
* We can do this with the help of revert command like

git revert a20e325

* With this command we can revert back and new commit will be created and our history will be preserved.
* Git reset we can only use when the changes are in local only, if our code is pushed to remote repo we cannot use reset command because it will effect the team members also.
* In that case we need to use the git revert command only.