git and GitHub have been one of the most popular tools used in today's fast

moving it World pretty much every big organization is using a distributed

Version Control System to support distributed non-linear workflows by providing data Assurance for developing

quality software multinational companies such as Google Facebook Microsoft

Twitter LinkedIn are using git and GitHub on a daily basis across various

Industries this proves that if you're someone looking forward to building your carrier in the domain then it is

definitely the goal in time for you hi all I welcome you to this full course by edureka and today we're going to talk

about one of the most South tools git and GitHub and what follows is hopefully everything you need to know to get

started with Git and GitHub but before we begin let's take a look at our agenda for today first of all we're going going

to start by introduction to get where we will discuss what is Version Control what is git and the git installation and

setup next we will be moving to how to use GitHub here we will discuss about what is GitHub and how to use it also we

will talk about some GitHub commands and the GitHub workflow followed by which we will talk about git branching and

stashing where we will discuss branching in git how to handle merge conflicts and get stashing next we will learn about

Kit merge and gitry base where you will learn the what is git rebase what is gith merge git merge versus get rebase

in the fifth module we will talk about git commands where you will get an overview of top git commands moving on

we're going to talk about Jenkins git integration where we will see what is Jenkins and why are Jenkins and get used

together followed by which we will talk about gitlab cicd pipeline where we'll

discuss about introduction to CI CD introduction to gitlab CI CD and also

building a CI CD pipeline using gitlab next we will learn about git Ops where you will learn what is git Ops and why

do we need it also how does git Ops work finally we will end this session with

the git interview questions where we'll discuss basic interview questions intermediate level interview questions

and advanced interview questions with this I come to the end of my agenda kindly take up this time to go ahead and

subscribe to us and do not forget to hit that Bell icon to never miss an update from the edureka YouTube channel also to

learn more about git do not forget to check out our certification training the link to which is given in the

description box below so without Much Ado let's get started

foreign so we'll try to learn what is Version

Control what is git what are the benefits of using the git then we will be going ahead with the git installation

and the setup along with basic operations and commands with this let's

start our session for today so let us try to understand what is a Version Control Suppose there are three developers in a

team who are working remotely from their own system and who need to contribute towards a project

they will be working on their own requirement they will be developing their own web pages

now how their day-to-day activities looks like say developer 1 will be

working onto the streaming Pages developer 2 will be writing the code on some user info Pages developer 3 will be

contributing towards the payment Pages now try to understand they are developing the code with respect to

their specific functionality at different interval of time now it is very important for them to collaborate

with each other so that they can save all of their application code as a

single source code now how does this particular system work how they will be

able to collaborate with each other considering that they have been working on different project considering they

have been working on different functionalities and they have been working remotely

this is where the Version Control System comes into the picture say for example

developer 1 started developing a code for a streaming page and he'll be

sharing that particular code with developer 2 who is then working on adding a user information page

further both of these particular pages will be shared to be collaborated by developer 3 who is the working onto the

payment information page now it is very important for all the three developers

to contribute together to form a complete application and this is what a

Version Control is all about so let's try to understand what is a Version Control Version Control is a kind of

system where you are recording all the changes that have been made to any single or multiple files over a period

of time later these versions can be referred which are stored into the

Repository to understand Version Control System there are three categories of a Version

Control System the first one is local Version Control System where all the

code all the changes are being tracked locally onto the developers system then

we have a centralized Version Control System where developers can't connect to a centralized server they can update

their code onto the centralized server and they can collaborate with each other

then comes a distributed Version Control System in distributed Version Control System each developer have their own

local copy of the repository which they can further collaborate with other developers by merging them by pushing

them into a remote repository we will further be discussing with the centralized version control system and

distributed version control system in the further slides so why do we need Version Control System

what is the use of having a Version Control System yes for better collaboration

since there are multiple developers who are contributing to the project hence it

is very important for them to collaborate with each other managing the versions as we have discussed that yes

all the changes that are made to the application code all the changes made to

any file will be tracked and hence we are tracking those particular changes with a new version associated with a new

change rollback functionality it is easy to roll back to the previous

stable version in case there is any issue with your current version reducing the downtime which can help in

reversing the faulty updates how it will help in reversing the faulty updates in a timely manner because we

will get to know that who has done that particular update what all changes has been made in that particular update

using a Version Control System we can identify the person we can identify the developer along with the changes made by

a developer and using that we can reverse those particular updates to reduce the downtime and last but not the

least we can analyze our project we can compare different version of our project at how we have progressed with different

versions of a code let's try to understand what is a repository a repository or commonly

known as repo is a kind of a storage place is a kind of a directory structure where you can save all the code

associated with your project it can either be available locally on any

developer's workstation or you can save all of your code in any remote

repository also the one of such example for a remote repository is a GitHub

which we will be learning in bit detail in the further slides you can keep any

kind of file into your repository be it a code file be it images be it a text

file or whatever is associated with respect to your code all of these

particular files can be saved into the repository further let's try to understand the difference between two

commonly used Version Control System One centralized and another one distributed

centralized version control system in centralized Version Control System we have a central server where the source

code is managed each developer connects to that particular Central server from

their own workstation update a specific code and commit that specific code into

the central server directly whereas in distributed Version Control System each

developer have a local copy of a code into their own workstation which is also

known as a local repository any updates or changes made into the code are first

committed to the local repository once the changes are committed to the local repository it is then synced to a

central server which is also called as a remote repository for further collaboration

now let's try to understand what is git git is a kind of a distributed Version

Control System where each developer each contributor can sync the code from a

remote repository they can create their own local repository they can make changes to their local repository and

collaborate that with the remote repository they can sync their updates

they can sync their changes first in the local repository and further collaborate

that with a remote repository whatever changes are available inside the remote

repository will be made available for the other developers who are contributing for the project

further let's try to understand some benefits associated with Git some basic

features of git economical as it has been released under

the GPL license it is free and open source that means it do not need any

licenses so there is no cost involved for using git non-linear git supports a

non-linear development or a parallel development which we will see as part of the demonstration when we will be

developing in parallel along the Deep branches snapshots git takes the snapshot get

tracked the changes that are made inside the file rather than just tracking the

file it helps you to track what all changes has been made in a specific code

file and by which developer further since it is a distributed kind

of a Version Control System every developer every contributor can have their own copy of the repository which

is stored locally on their Workstation speed is one of the another important feature of Kit where every change that

you are saving is done at a lightning fast speed robustness nearly every task in git is

undoable if you are making any change in a file if you are saving those

particular changes as your tracking system as your version control system it

is easy for you to roll back those particular changes and to undo those particular changes

Integrity no change in git can be made without git recording it that means

whatever changes you are doing in your code irrespective of whatever file it is

all those changes will be dragged into the kit and last but not the least the

most important part it supports the parallel development it helps contributors to work into

different branches so they can collaborate together at a later stage

let us now try to understand that how git actually works so it is very

important for us to understand a basic workflow of git this is how a basic

workflow of git looks like where we have three different logical areas within kit

working copy staging area and local repository whenever A change is done by

a developer those particular changes said to be present into the working copy

or which is also known as a working directory whenever changes are ready to

be committed they are then moved from working copy to a staging area which is

also called as a temporary location which is present inside a git before it

can actually be committed to the local Repository all the committed changes

present in a repository are stored into the local repository this is how a local

operations of Kit looks like further the local repository can be

synced with the remote repository which is available over the internet you can

push your code from local repository to remote repository you can even clone the

code from your remote repository to local repository and this is how a code

or different developers contributes towards a common goal let's try to

understand this particular workflow bit more in detail with this workflow we

have a working directory available using the git add command any changes that has

been made by a developer will be moved from a working directory logical area to

a staging area this is what you call as a intermediate location this is what is also known as a

temporary location once all the changes which are present in staging area are ready to be saved

are ready to be committed they can then be committed into the local repository

once you commit all of your update they will be updated they will be saved into

the local repository when you need to sync your local repository with a remote

repository you can use a git push command to sync the code from your local

repository to the remote repository and that is how you can share your specific

code update with other Developers there is a different scenario where other

developers will be contributing to the same remote Repository you can get the latest copy of the code

from remote repository into your local repository using a git pull operation

once the code is pulled from remote repository to local repository developer

can then work on to their specific code changes and later merge those specific

changes into their local repository this is how a basic workflow looks like for

kit let's try to go ahead with the git installation and setup

for get installation and setup you need to have a git bash for your windows how

can you install a git bash in the Windows operating system for all the users who want to use git bash in the

Windows operating system you can go to the download site for git search for

your appropriate operating system if you wish to download that for Windows click

onto the download for Windows button and there an executable file will be downloaded on your local system which

you can use to install the git bash let me show you from browser how this looks

like this is how a download page will look like where you can choose different

operating system with respect to your requirement whether you want to install it on Mac OS windows or a Linux based

operating system Windows user can click on download button so that it can

download the executable with respect to your Windows operating system which you

can further execute to install the get bash on your respective operating system for Linux users you would further need

to install the git in the Linux environment using a package manager we

will be giving a quick demonstration how you can install git on your Ubuntu operating system well we will be using

an apt package manager to install the git in your environment

so let's switch towards some hands-on experience so to save time I have already set up a

virtual machine which is based out of an Ubuntu operating system where now we will be installing git so how you can

install git we can install it using a package manager the package manager

associated with Ubuntu operating system is your apt let's first update our

package manager by a command called sudo apt update to update the package manager

you need to have the root privileges now since I am not a root user that is where

I need to use a sudo operation which will help me to update all of my

packages that are available in this Ubuntu operating system once all the packages are updated for my

Ubuntu operating system I can clear my screen and further you

can use a command called sudo app install and the name of the package that

you wish to install in our case it is git once you will proceed with this

particular command it will further download a git package in your environment since this package is already available

in my operating system hence it has not installed this particular package once

again how we can ensure that this package is available and it is ready to

be used by executing a command called Git Version using a Git Version command I can see

that yes a Git Version 2.17.1 is already installed in my system

and yes we are good to go ahead with the usage and the basic operational commands

of the kit let's try to understand basic git operations and commands this is how

we have categorized some basic and commonly used git commands for you where

for creating a repository we can use a get init command which can initialize a

repository from a beginning from a scratch further once the repository is

created you can make changes to a specific code you can add those particular changes from working location

from working directory to staging area and you can further commit those particular changes from the staging area

to your local Repository since we have been discussing about a lot for parallel

development so yes git can be used to create branches where different

contributors can work onto the different branches where a developer can create multiple branches with respect to

multiple requirements and he can further merge those particular changes into the

master Branch once the changes are finalized we will also see that how

rebase can actually help you to track all the changes in a sequential manner

once the changes are made into the local repository this is how using the sync

repository commands we can further sync the data we can further sync the changes

associated with respect to our code into a remote repository using our git push

and git pull operation let's go ahead and try to see what all commands we can use to create the

Repository as we have seen git init is the command which can be used to create a new git

repository on your local machine and yes since it is something you are creating

on a local machine it is also called as a local repository git clone is used to

clone any existing remote repository from a remote location to your local machine whereas get Fork can be used to

create a copy of that particular remote repository into your GitHub account directly the difference between git

clone and git Fork over here is with Git clone you are creating a copy of a

remote repository on a local machine with Git Fork you can create a copy of a

original repository which is available in a remote location directly to your GitHub account

now let's try to understand basic git operational commands for syncing the repositories

first of all you will need to add a remote repository with your local Repository with get remote add command

you can add any remote repository you can link that particular remote repository with your local repository

which will further be used to sync the code from your remote repository to

local repository using a git pull command and from your local repository

to remote repository using a git push command all of these commands we will be

covering at a later stage when we will be going to our demonstration now let's go ahead and try to understand

few other commands which will make sense when we will actually be doing a demonstration so let's go to our

terminal and clear our screen let's create our application code

directory where all the code will be saved with respect to my application so using an mkdir command I am creating a

demo app folder where I'll be saving all of my code using a CD command I am changing my

directory to the demo app I'll just confirm what is my present working directory which says yes I am

present in the demo app folder where I'll be saving all the changes made to

my application code before git can start tracking any changes that you are making in this

particular location you will first need to initialize the repository and as we

have discussed earlier how we can initialize our repository using a get init command

what get init command does for us is it create or it initialize an empty git

repository for us within this particular location as we can see that a DOT kit

folder has been created in my present working directory which means that yes

my git has been initialized or my local repository has been initialized let me

clear this particular screen once the local repository has been initialized we will now configure our

gate to identify the user and the respective email address so that any

changes which has been made in my application code can be tracked against the developer's name we will be able to

identify the contributor that who has contributed towards that particular change or who has done those particular

changes so let's configure our repository with Git config command and

setting up some Global parameters the global parameters which we will be

setting up is user.name where we will need to specify the name of the

developer who is contributing towards this particular code say I will choose a

name I am devops trainer once you have configured a username

parameter you would also need to specify you will also need to configure an email

address associated with that particular user so let's configure another Global

parameter which is your user email and this is where I'll be specifying my

email address so that if anybody needs to communicate with me if anybody needs

to discuss that what our changes has been done why these particular changes has been done I can be contacted on this

particular email address all of this is recorded when you commit your changes

and that is something which we will see during our demonstration once git is configured now let's start

working with our code file and start tracking our changes let's clear this particular screen

the first thing that we will be learning one of the first command that we will be learning about git is git status

get status command actually helps you to highlight the status of your repository

how your local repository looks like since we have not committed anything in

our local repository yes so our Master branch do not have any comments yet now

please note Master is the default branch which gets initialized which gets created whenever you initialize a local

Repository so now we will write our first file our first application code

file edureka1.txt so let's use our VI Editor to create our

first code file which is edureka1.txt with the vi editor you can add any

command like hi this is a session from edurica

we can further save this particular file which will act as a code file for us

once a code file is added in your repository let's see how git track this

particular code file how you can check the status of get using a git status

command using a git status yes we can highlight that in my master Branch there

are no commits that has been saved yet into my local repository however the

file which we have just created the file edureka1.txt which we have just created

in our file system has been referred as a untracked file which means yes this

particular file is an untracked file git doesn't know anything about this

particular file this is where it is said to be present in a working directory I

hope everybody remembers that git Works into the three logical areas working

directory staging area and local Repository all the edited files will be

present in your working directory and how we can move our application code

file from a working directory to a staging area using a command called git

add using a command called git add and the name of the file we can add that

particular file from our working directory to the staging area

so let's use git add command onto our edureka1.txt and let's see how our git

status looks like now my git status looks like that yes it

is now a tracked file although we do not have any commits yet done into my local

repository but we still have a tracked file a new file has been tracked by my

get local Repository but this is not yet committed to my local repository and it is still

available inside the staging area we have just staged this particular file

before it can be committed now let's see how we can commit this particular file

into our local Repository we can commit this particular file into

our local repository using a git commit command and all the necessary options

the parameter the options that you need to pass along with get commit is the

message is the commit message associated with respect to that commit and that is

what you specify with hyphen M as an option so let's say my first message or

my message is committing my first file

eureka1.txt

and this is how the output of a commit command will look like where you should

be able to see that yes a commit ID is generated along with the commit message

and you will be able to see that what all changes has been done in a summary

where we have changed one file where one insertion has been done in a particular

file and yes this particular file has been created in my local Repository now once

the file is committed into the local repository let's see how my git status

looks like with this git status we can see we do

not have any unsaved changes that means whatever changes we have done in all of

our application code file they all are committed and they all are saved into

the local Repository now let us try to understand that how we can track the

changes made to our local repository how we can see the summary of commits what

has been made into our local repository for that we can use a command called git

log with a git log command git will actually present the output of all the

commit messages that you have made inside your repository now let's try to

understand this particular command bit more in detail as an output we can see that yes a hash ID has been generated

for a specific commit where the author who has made that particular commit who

has made those particular changes in the application code file is I am devops trainer and the associated email address

with that particular user is I am devops trainer gmail.com now we want to check

what our changes has been made in my system with this commitment message we

can check the details with respect to any commit message using a git show command and a end that with the commit

ID you can just specify initial 7 to 8 characters of your commit ID which will

be able to identify a specific commit hash ID as all the commit IDs are

different let us go ahead and clear our screen and let's try to re-edit our

edureka1.txt to make an update into our application code again I'll be using a

VI editor on my file edureka1.txt where I can add more number

of lines into my application code I can make changes into my application file

and yes further we will be tracking all these particular changes into the local

repository say for example I just want to replace this uppercase t with a

lowercase t so yes I can update my code and this is where the first change has

been made as part of a second change which I am making in this particular file I am

adding a new line so please note how many changes we have

done let's save these particular changes and see how our git looks like once you

have updated your edureka1.txt code file using a git status command once again this time we

will be able to see that yes some changes has been done in edureka1.txt

this file has been modified now and I hope everybody can see that it is

not an untracked file anymore why not untrack because this is already being

tracked by your Git it is already being tracked by your local repository but yes

it is present inside your working directory why because we have edited

this particular file hence this particular file is present in a working directory how we can move this

particular file from a working directory to a staging area I hope everybody knows by now using a git add and edureka1.txt

now if you want to add multiple files from working directory to staging area

you can even use a DOT operator which can add all the files that are available

in the working directory and move that to a staging area so using a DOT operator it can even add

multiple files to the staging area let's see how our repository looks like when we have added this particular file from

working directory to staging area yes this modified file is now available

in my staging area which is now ready to be committed and yes with this commit a

new commit ID has been created please note the commit ID it's 0 f a okay a new

commit ID has been created where it says that one file has been changed

three lines has been inserted and one line has been deleted but we haven't

deleted any line if you will try to remember what all changes we have done in edureka1.txt we have added three

changes so let's see how our commit message looks like what do we mean by

one deletion which is being presented as the get commit message once again we can

clear this particular screen and using a git log command we can track what all

commits has been done in my local Repository with this git log we can now see that

this two commits has been done and the latest commit which starts with the

commit ID 0fa is associated with the update that I have made in my

edureka1.txt so if I need to check more details with respect to this particular commit I can use this particular commit

ID and I can display the result I can show more details with respect to this

particular commit ID and see what all changes has been done as part of this

commit ID now please note we have still done the change in our file at eureka1.txt where

the deletion part is highlighted by a hyphen or a minus sign and an addition

is shown with a plus sign so as you can see that whatever changes

we have made in any line git removes the existing line if you are making me

changes in that particular line git removes that particular line and save a new line with the updates

that you have made in that particular file this is where one deletion has come

from where the original line was deleted why because we have made the change in

that particular line so original line was removed a new line was saved along with a blank line and

the third line where we have added a new line and this is how your git commit

message looks like now since our local repository is set up let's see what all files is dragged by

my get local repository for that we can use a command called kit LS files and

using git LS file commands it will list out all the files that are being tracked by your Repository let's see what if we

will add a new file called edureka2.txt say I can just touch a

blank file called edureka2.txt on my present working directory on my code directory which is

being tracked by git and let's see how my file system looks like so yes I can

see that I have two files present in my file system at eureka1.txt and edureka2.txt

let's see how my get status looks like in git status yes I have one unsaved

change which says edureka2.txt is a kind of an untracked

file right because this file has been added as a new file that is why it is a

untracked file and let's see if it is being tracked by my git or not and how

we can check that we can check that using git LS files command as you can see the output still my git

is tracking only a single file which is edureka1.txt what if I need to add my

edureka2.txt also in local repository once again we will be following with the

same operation where I can add my git edureka2.txt which will further move that particular

file into the staging area and from the staging area I can commit that particular file using a commit message

with committing my second file at eureka2.txt

yes with this commit we have added one file where no line was inserted or no

line was deleted why because it was a blank file so if I clear quickly

checking onto the get status it is clean checking on to the list of files that

are tracked by my get so yes now both the files are tracked by my get into the local repository and using the git log

command I can see that yes there are three commets made in my local repository Master branch

now as we are able to set up our local repository as we have tracked all of our

changes into the local repository let's try to learn more about how we can sync

these particular changes to any remote repository for that you need to have a

remote repository available where you can sync all these particular changes from your local repository to a remote

repository now as part of an example for this particular session I already have a

remote repository created in a GitHub account and this is a URL of my remote

repository I will copy this particular remote repository link

and Link it with my local repository I hope everybody remember that when we

have discussed the link between the local repository and the remote repository we can make that particular

link using a git remote add command

we can add or link any remote repository with our local repository using a

command called git remote add and providing an alias to your repository by

default the Las will be used as a origin we can use a git remote add origin

command to add your remote repository and Link it with your local repository

with this particular command it will help you to link any remote repository

with your local Repository how we can check whether a remote repository has

been linked with our local repository or Not by using a git remote hyphen V

command hyphen V option is used to print out the repository location in a verbose

mode where we can see from the output that origin Alias is linked with my

remote repository which I have created for this particular demonstration so yes our remote repository has been linked

with the local Repository next we need to now sync our code from local

repository to remote repository and that is where we will be learning about a new command which is git push command now

git push actually helps you to push all the changes from your local repository

to the remote Repository and what are arguments you need to pass with Git push you will first need to pass the alias

the Alias of the remote repository where you want to sync or where you want to

push these particular changes and the branch for which this particular

changes needs to be pushed I hope everybody remember that the default Branch with which we are working so far

is the master Branch so yes what we need to push finally we need to push our

local repository changes to the origin remote repository along with the master

branch and yes when we will try to push these particular changes it will ask you

to authenticate to your GitHub account whosoever is the owner of this

particular Repository since I am the owner of this particular repository I will be providing my own credentials

and with the successful credentials it will be able to push all the changes

that were present that were tracked in my local repository they will be pushed

to the remote repository so let's go to our remote repository and see how it

looks like after the code is synced from a local repository to remote Repository

let me go to the address bar and quickly refresh the page where I can now see

that yes inside my remote repository I have two files available where

edureka1.txt was the first file which we have created and this is how the file

looks like on the developers workstation on my workstation and where the

edureka2.txt is a kind of a blank file but still committed to this particular

repository and who has done all these changes all these changes has been done

by I am devops trainer all the details associated with respect to your local

repository are synced to your remote repository whatever number of commits we

had in the local Repository we have three commits available in the

local repository that all is synced into my remote repository under the commits

section so if I want to check all the comments associated with respect to my repository

this is how I can track all the thumbnails and if you want to track the commits ID you will be able to see

whatever commits were available inside my local repository they all are sent to

the remote Repository this is how we can sync a data we can

sync our code we can sync the changes from a local repository to our remote Repository now let us try to understand

bit more commands which are associated with the parallel development and which

is once again one of the most important topic for advanced topic when you are working with the kit now first important

topic when you are working with the parallel development in gate is branching branching isn't very important

and integral part of any Version Control System why because we are doing a

non-linear kind of a development unlike any other version control system git do

not actually create any copy of the existing file whereas it will just point to the

respective snapshot when you have actually created a branch so let's see how we can actually create

a branch from any existing branch in my example I am presently connected

to a Master Branch how I can ensure that using a command called git branch

now git Branch will list out all the branches that are available inside your

local repository with the one highlighted with the asterisk is the one

which is active currently you can also check to which branch you

are connected by using a command called git status with a git status command it will

highlight you on which branch are you connected to so presently I am connected

to a Master branch and there are no pending commits there are no pending

changes available inside my master branch this is now we will be using the master

Branch to create our new Branch so how do we create our new Branch using a

command called git branch and an argument Branch name or name of

the branch which we want to create say for example I want to create a new

branch called edureka branch with this what git has done git has

created a new Branch for me with the name called edureka Branch let's see how

this particular Branch looks like so first thing if I'll check what all branches do I have in my repository so

yes I can see that there are two branches available now in my local

repository but my gate is pointing to my master branch

now why do you actually need a branching in git or why do you actually need

branching and development because whenever you are making any updates

whenever you are making any changes in your respective code file you work with

any such requirement in a separate Branch so that you do not touch your

master Branch or we are not making any changes to the master branch Master

branch is said to be used for the production specific code or for all the

application code that has been approved for the production whatever testing code

you need to update you should create a branch associated with that and then

make changes into the respective branch so now we have created a branch called

edureka Branch let's switch to this particular branch and see how we can do

a parallel development using the branching feature so how to switch to a different branch

using a command called git checkout and the name of the branch in our case the

name of the branch is edureka branch which means that yes I am switching from

Master Branch to edureka Branch if I will clear my screen

and check what all branches are there in my system and where my git is connected

to so yes you will be able to see the asterisk has been moved from my master Branch to the edureka branch right now

which means yes now I am connected to my edureka branch and I can use this

particular Branch to make an update to make changes into my specific application code

let's see how the status of this particular Branch looks like using a git status command I can check the status

and this time I hope everybody noticed that yes it is showing the status of my

edureka branch right where there are no commits to be done there are no pending changes inside

the working directory or staging area everything associated with respect to

this particular branch is already saved or present in the local Repository

now the important point for us to understand whenever a branch is created

or when we have created this particular edureka branch it will have all the commits available

from the master that were present in my master branch that means

when we create a new Branch from any existing Branch all the commits ID

associated with the source Branch are copied to the new branch that we have

just created that means if we will try to understand it in this scenario we have created our

edureka Branch from source as master that is where whatever commits were

available in my master Branch they all will be replicated inside my edureka

branch as we can confirm that yes now we are connected to the git Branch let's

see what all commit messages are available inside this particular Branch using a git lock command we can display

all the commit messages that are available inside this particular branch and yes we can see all the three commits

that were part of my master Branch are also now available in my edureka branch

that means we can proceed with our parallel development where we can make

the changes inside the edureka branch as well as inside the master Branch parallel and there are two different

branches where parallel development can be worked upon now let's try to understand how parallel

development works with the branching let's create a new commit let's update a

file inside my Branch edureka branch once again quickly checking on to the

status I am still connected to my edureka Branch where there are no pending changes available inside the

working directory or staging area let us create a new file called

Branch 1 dot txt file where we will add some text to this particular code

this file is created in branch

at Eureka branch and we will save this particular file now once again try and understand we

have created this particular Branch one dot txt in our edureka Branch so if we

will check the status of Git it will say that yes it is connected to the edureka

branch and since we have added this particular file since a new file has been added into the file system

it is a kind of an untracked file which is not tracked by my git so to track that particular file we will

again add this particular branch 1.txt and once it is added we will commit this

particular file to our local Repository

committing file branch1.txt with this commit please note the commit

is being done into my edureka Branch let me clear the screen and quickly check on

to the get status yes the get status looks clean we do not have any unsaved

changes quickly checking onto the get log which will show me that yes our Master

Branch just have three commits whereas the number of commits in my edureka Branch are four and how we can quickly

check onto the number of log commits that are present in my edureka Branch by using a git log and the branch name

so with the git log on to the edureka branch I can see that yes it has four commands where we can now see that yes

we are proceeding with the parallel development where there are just three commits done in master and four commits

are available in my EDU breaker branch but once you have made changes into any

new branch which you have created for a new feature you will finally need to merge those particular changes back to

your master branch and how we can merge those particular changes into my master Branch by using a

git merge command so there are two different ways how we can actually merge

changes made into one branch into the another Branch by using a git merge and

get rebase so here we will be focusing more on to the git merge so quickly checking on

which branch I am connected to I am presently connected to my master Branch the changes with respect to my Branch 1

dot txt was made in my edureka branch that means if I'll try to check my local

repository on my master Branch it will not show me

branch1.txt whereas if I will check out to my edureka branch

and I'll check the list of files tracked by my local repository I should be able

to see branch1.txt which is tracked inside the edureka branch for me

how I can merge these particular changes how I can merge the code how I can merge

the changes done in my edureka Branch to master I'll be checking out to my master

Branch using a git checkout Master I'll confirm that I am connected to a Master branch and using a git merge command on

my Branch name which I want to merge I will be able to merge those changes from

edureka Branch to my master using git merge edureka Branch command I should be

able to merge all the new changes that were available in my edureka Branch to

my master branch and this is where we can see from the output that yes it has

fast forward our Master Branch where file called branch1.txt was added inside

my master if I'll quickly clear my screen and check onto the git status my git

status looks clean I am connected to master and there are no unsaved changes

on my massive branch however if I'll now check the logs available in my master

Branch I can confirm that yes the fourth commit that was done in the edureka

branch is something which is now available inside the master Branch also

and if we will track the list of files that are managed by my local repository

I can confirm that yes branch 1.txt which was the change done in the edureka

branch is now available inside my master Branch also this is how we work with the parallel

development on working with the branching and merging those particular Branch after the changes being done

as we have discussed that yes parallel development can go ahead with different branches and once the merge is being

done the changes made into the new Branch can be merged into your master

Branch using a git merge command further as part of a parallel development there

is an additional concept of rebasing where you can even rebase your branches

from any existing Branch say for example we have created a new Branch from Master

at C2 then there is a new commit done in the master at C3 and an additional commit

done into a branch called C4 commit then when you will rebase that specific

Branch it will look like that you have made all these changes sequentially

where the change done by our Master Branch will be incorporated in your new

Branch where we are rebasing my branch and this is what a rebase command can be

used for and let's try to see it with the demonstration how rebase is done

let's clear our screen let's ensure we are connected to the master Branch let's

check what all log messages are available inside my master branch and

create a new branch which we will be using for the rebase demonstration

using a git Branch command I'll be creating a new branch called a rebase

branch and yes using a git checkout I'll be

able to check out to my rebase Branch where all the four commits that are

associated with my master are also available in the rebase branch

now with rebase again it is a kind of a branch where we can proceed with the

parallel development say for example we are updating our first file called edureka1.txt

inside my rebase branch and I am adding a new line in vbase branch

that is the change which we are making into our rebase branch finally I'll be

committing that particular change by adding that into a staging area and further committing it with a commit

message well I have updated a eureka1.txt in

rebase branch so yes this is how now my git commit logs look like where I have fifth log

available inside my rebase branch now this git log output is overwhelming

so there is an option available with Git log where we can restrict the output to

a single line and the command with that is git log one

line so using one line we can see what all commits are there and the associated

commit message this is how my commit looks like into my rebase branch

if I want to check how my commit looks like into the master Branch I can just specify the branch name and this is how

my comments looks like into the master Branch please note a new commit E64 is

done into the rebase branch whereas my master branch is still at 2 e d e

now let's check out to our Master branch and make a new commit inside my master

Branch so as part of a new commit let me create a new file which is called at

eureka3.txt I am adding this particular file

after the rebase branch is created this is where I can add this particular

file from my working directory to the staging area and I can make a commit to this particular file in my master branch

so I am adding a Eureka 3 dot txt in my master Branch now please note what we

have done so far we created a rebase branch from the master at Point 2 e d e

after that we have made a commit in a rebase branch and a commit is made in

the master Branch Also let's see how our logs look like now so let's check onto

the git logs of my master branch and this is how a git log in the master looks like let's see how the git log of

the rebase branch looks like so if we will check the git log of the rebase branch this is how the output of my

rebase Branch looks like which means that yes we are doing a parallel development

in my master branch and parallel development in my vbase branch

now how a rebase is actually used for Freebase will actually update the base

of your branch now let us try to understand what is a base of a branch when we created a rebase branch the base

of my master was present at 2 ede

whereas master has now moved from 2 e d e to 84 B7

this is where the master has been updated and now we need to update the

base of my rebase Branch from the old Master to the new Master

how we can update that using a git rebase command so we'll be checking out

to our rebase branch we will ensure that yes we are connected to our rebase

branch we can see that yes asterisk is pointing to the rebase branch and using

a git rebase command along with the source Branch with which we want to

rebase it will update my master Branch base to

my rebase branch earlier my log commits look like this

before the rebase where the base of Master was at 2 ede

but after the rebase is done let us try to check how my git logs look like for

my rebase branch here the base of my branch is now

updated to a new master that is whatever commits has been done inside my master

after the branch was created they all has been rebased into my Freebase branch

let us now try to understand it from this particular diagram say for example at C2 commit we created our new branch

and after C2 we have added a C3 commit in master and C4 commit in my new Branch

once we will proceed with the rebase it will proceed with a sequential commit log messages where the new commit which

was done in the master will become the base of my new Branch commit earlier the

base of my new Branch C4 was at C2 but

after rebase my C4 has a base of C3

with this I hope everybody is able to understand the basic operation of the vbase command let us now try to

understand few bonus command which you should definitely keep in mind when you are working with the git as a tool where

you can archive your repositories yes you can use a git archive command to

actually archive any branch from your local Repository and you can create a compressed file

which you can further share it with other developers or other project members we also have an option to create

a bundle out of your particular repository and yes this bundle can actually help you to be created from a

specific branch using a git bundle command add a create operation you can create a new bundle of

your master branch and last but not the least one of the most important command

one of the most important operation that kit offers you is the git stash command

which is used to save all of your uncommitted changes into a temporary

location [Music] thank you how to use GitHub so firstly

we will understand why we need GitHub and what it really is then we will learn to create our very own first repository

we will also understand what branches are and how to create them then we will

further discuss a few basic GitHub operations the first one being the commit operation followed by the pull

operation finally we will end the session with the case study on how Microsoft uses GitHub now let's move on

to the first part of today's session the need for GitHub it is extremely important for software developers to

work on a web-based platform to share their projects and collaborate with other developers this platform must be a

version control system that is it must enable multiple people to simultaneously work on a single project each person

edits his or her own copy of the files and chooses when to share those changes

with the rest of the team this application must also be capable of Hosting millions of programmers and

hobbyists that download and evaluate each other's work GitHub is one such platform of choice for developers that

can host multiple programmers and review their code GitHub has several competitors for

instance git lab gitlab is an open source web interface and Source control

platform based on git whereas Microsoft team Foundation server is an Enterprise

grade server for teams to share code track work and ship software for any

language all in a single package bitbucket on the other hand stores all

of your git and Mercurial source code in one place with unlimited private repositories

so what really makes GitHub so powerful and popular among Developers GitHub is an open source platform and

the community is really what fuels it moreover GitHub is the platform of

choice for developers from various large corporations too Microsoft is the number

one contributor to the system but there are also Google sap Airbnb IBM PayPal

and many others exposure and insight that you can get on GitHub are simply

unmatched by any other platform here you can discover code written by others learn from it and even use it for your

own projects versions control on GitHub works very much like Microsoft Office or Google

drive it simply tracks all the changes made to your code and who makes them you

can always review the detailed change log that neatly hosts all of the relevant information using GitHub

eliminates the need for complex corporate security solution because everything is on cloud the platforms

protects code branches verifies commit signing and controls access

now that we know why we need GitHub let us understand what is GitHub GitHub is a

git repository hosting service that provides a web-based graphical interface with many features a repository is

usually used to organize a single project repositories can contain folders files images videos spreadsheets

anything your project needs let's say for example a team wants to work on a

particular project here they can simultaneously write and update the code to a central repository which is present

on GitHub so GitHub is a highly used software that is typically used for

Version Control it is helpful when more than just see one person is working on a

project for example a software development team wants to build a website and everyone has to update their

codes simultaneously while working on this project in this case GitHub helps

them to build a centralized repository where everyone can upload edit and manage the code files

most software projects have a bug tracker of some kind github's tracker is

called issues and has its very own section in every repository issues

basically are a great way to keep track of tasks enhancements and bugs for your project

moving on people often get confused between the terms git and GitHub now let

me clearly explain the difference between them git is simply a version control system

that lets you manage and track changes within your project whereas GitHub is a

cloud-based service that lets you manage git repositories so basically git is the tool and GitHub is the service

now that we know the difference between git and GitHub let us move on and understand how these two work hand in

hand we already know that git is a Version Control tool that will allow you to

perform all kinds of operations to fetch data from the central server or push data to it whereas GitHub is a code

hosting platform for Version Control collaboration GitHub is basically a company that

allows you to host a central repository in a remote server now without any

further Ado let's get started with the demonstration on how to use GitHub so

for this demonstration we're working on the website version of GitHub there's another version of GitHub that is the

desktop version which you can download it to your personal computer so we're simply going to search for

GitHub in our search engine the first link will lead you to the official website of GitHub so I'm going

to click on that so this will redirect me to the main homepage of GitHub as you can see there

is a search GitHub option there are also two buttons that says sign in and sign up if you're new to GitHub you can

simply enter in your credentials that is a username email password and sign up for GitHub but if you already have an

account like I do I'm simply going to click on the sign in button and it'll redirect me to a page where I have to

enter the credentials that is my email address and password I'm going to do that now and I'm going to click on the

sign in button now this is the main page of my account as you can see I have no

repositories it's all new it's all fresh but if you're not new to GitHub you can view all of your repositories on the

left hand corner now before we move on I'm going to explain you all the features that are present within GitHub

so you can see a search bar here so the search bar will allow you to look for profiles certain keywords look for

different kinds of projects that are available on GitHub all of those can be done using this bar here and you can see

four options next to the bar that says pull request issues Marketplace explore pull requests will learn later on in

this session but the issues in Marketplace we won't be discussing in this video for now the explore button on

the other hand is an extremely important and interesting button so once I click on that it'll redirect me to a page with

some activities that are going on around in GitHub you can see here the trending repositories they're attending

developers basically this is a fee that will allow you to interact with developers and other people

collaborators from all around the world basically in Instagram too you have an explore button which will allow you to

interact with different people from all around the world so the same concept is implied in the GitHub explore button too

so you can explore top picks you can explore tending repositories developers

basically it's an interaction with other people from different parts of the world so I hope that's clear now the most

important part of the session are the three buttons that are available in the right hand corner of the navigation bar

so you can see there's a bell icon there's a plus icon there's a pixelated icon on the right hand corner so the

Bell icon allows you to read notifications of your activities that occur in GitHub so that's what it really

is you can see the inbox will allow you to view all of the notifications you can also view the Android notifications by

clicking on this unread button as of now I don't have any notifications so there's nothing available you can also

group these notifications by the date or repository by clicking on this group by button here you can also view your save

notifications by clicking on here and the done button on the other hand will let you mark all of your notifications

that you're done with your previous notifications so these are the three important buttons you have to know in

this Bell icon and the filters are not necessary as of now so I'm not going to discuss that this button on the other

hand will allow you to manage node your notification settings and your subscriptions too so that's all for this

Bell icon the next important button is this plus icon as you can see there are five drop down options that appear here

the first one being new repository followed by import repository new just new organization new project so new

repository we've already discussed previously in this session a repository is a place where you create your files

for your project it's basically a storage space right so you repository can directly interact with your git

right so the new repository option will allow you to make files a repository to

your GitHub account right the git on the other hand the tool that which we use to make local repositories in a personal

computer can be directly pushed on the local repositories can directly be pushed onto your GitHub account so

that's what the new repository pattern allows you to do so but the new project

on the other hand is a place to track issues features and other tasks that are

related to the code within the repository you can also connect with the devops build and deploy process assign

people to tasks and so on by using this button that is the new project button so the difference between the new

repository button and the new project button is that projects in GitHub are only a part of GitHub but not get but

the new repository option is a part of GitHub and git so that's the main

difference between new repository and new project I hope that's clear so the next button will drop me down some

interactions that I can make with my profile so if I click on this your profile option it will redirect me to a

page where I can edit my profile I can really create my identity using this

page so here if I click the edit profile I can add a bio about myself I can add the company in which I'm working in the

location at where I am the website Twitter username Etc all of that I can add here all of the information about

myself I can also view the repositories I'm working on currently or the repositories I've worked in the past

projects that I'm working on the packages and the entire contributions I've been making on GitHub from the last

year so basically it allows me to build an identity or it'll help me build my

profile on GitHub so I hope that's clear now if I click on this button and if I

want to sign out from a profile I can simply scroll down and click on the sign out button here and this will sign me

out of my account so that's all for getting started with GitHub these are the basics on what GitHub is and what

each of the button and options really do now if I want to move back to the main page of my GitHub profile I can simply

click on this octocat that's github's logo so I'm just going to click on this octocat logo and here I'm back to my

main page now before we move on and work on the different operations and options within

GitHub and learn different things about GitHub I'm going to give you a brief

overview on how to download the desktop version of GitHub so I'm simply going to

search for GitHub desktop on my search engine and I'm going to click on the first link that's available on this page

now I can simply click on this button that says download for Windows 64-bit that's compatible to my current version

of my personal computer if you have a Mac you can simply click on the Mac version and download it to your desktop

but as I've already mentioned previously that we're going to work on the website version so I'm going to Simply switch

back to this now let's quickly move on to the next part of the session create a repository so firstly let us

understand what a repository is it is simply a storage space for the correct

project that you're working on GitHub is a very popular Central repository that allows you to share your files whereas

git allows you to create local repositories that are present on the system you are working on

so you can basically push your local repository into GitHub and share it with other collaborators via the central one

now that we know what a repository is and how it works let's go on to the demonstration part and create a first

Repository so you can do this in two ways either you can click on your create repository

button that is present on the left side or you can as I've already mentioned in the previous part of the session you can

click on to this plus icon and you can click on the new repository option so this will redirect you to a page that

says create a new repository you can add your repository name I'm going to name my repository as edureka and it's

available all of your repository names must be unique from one another to identify them easily you can also add a

description which is optional I'm just going to add the description this is my

first apostrophe and a description allows people or other collaborators to understand what your repository is all

about but as a good developer or a good programmer you would definitely want to add a description and give an overview

of what your repository is all about there are two options now available that says private or public now you can

choose your repository to either be public or private so the private One lets you decide who can access your

profile whereas the public One lets anyone View and access your repository but you can choose who can commit to it

that's the difference between public and private repository I'm going to let my repository be public as of now now if

you scroll down you can see that you can initialize your repository with three options the first one being add a readme

file the second one being add a git ignore file you can always choose a add

a readme text file to your project which often contains information about the project and other necessary details the

user must be aware of when he or she is accessing that particular project now I

want a readme file for my repository so I'm going to click on this button here that's going to check it the next option

is add a DOT get ignore file so this file will let you ignore a list of files

when the user is pushing files to GitHub that's what this option really does but

I'm going to let this be unchecked for now for your repository to truly be open source you will need to license it so

others are free to use change and distribute the software you can simply click on choose a license option and

pick your the required license for your project there are several licenses like MIT GPL

Apache License 2.0 BSD Etc but for this repository we don't really need a

license so I'm going to untick this too and now you can see there's a piece of information that says this will set

master as the default Branch but I'm going to ignore this for now I'm going to explain about branches later on in

this session so this is all you have to do to create your first new repository you add a name you choose a description

you add an optional description you let your repository be either public or private and you initialize a repository

with either of these three options and I'm simply going to click on my create repository option now this will redirect

me to a page with all the information and the files that are currently present in my repository you can see here my

repository name is present here with the optional description that I gave and the

number of files currently we have only just one file that's the readme text file and that's present here

so this is all we have congratulations you just created your first repository now you can see there are some options

that says issues pull request actions projects Wiki security Etc we don't have

to really talk about all of these right now we will just learn about one option that says code here so this is really

important if you click on this button you can see that there's a link that is available here and https link so if you

copy this link and paste it on your git terminal that's present on your computer you can download this entire project

directly to your local system so that's what the link is for I hope that's clear and the next option that says open with

GitHub desktop will allow you to open this entire repository in your GitHub desktop version and you can also the

last option that says download zip will allow you to download this entire repository in the form of zip files so

all of your project files will be within that zip file so that's all you have to really know about your repository and

and I'm going to click on the readme text file that's available it will take me to another page with

some extra information about that file you can see currently we have two lines and the memory space that is allocated

to this file so we currently have two lines that is edu Recon this is my first repository and you can also see the

number of contributors to this project that is just one that's just me for now and you can view the history of the

commits or the changes that have been performed in your file so we'll come back to that part later on this session

but you can move back to your main page of this repository by clicking on the name button here so edureka is the name

of my repository so I'm going to click on that so now I'm back to the main page of my Repository

before we learn how to create our first Branch let us understand what branches are branches allow you to work on other

features that can be included and merged with the master Branch if required so what is the master Branch the master

branch is the main branch where your project resides on so all of the changes all of the activities that you do with

your main project lies or is on your default branch that is named as the master Branch so what really GitHub

allows you to do is it allows you to create additional branches so on these additional branches you can work on the

other features or you can experiment with your project and if you're happy with this you can simply merge these

features to your main branch that is your master Branch this is what branches are really for so they simply allow you

to work on other features that's what branches are so let's move on to the demonstration part and look at how we

can create our own branches so now if you look on the left corner you can see a button that says Master So currently

we're on the master to branch and there's only one branch and the master branches have already mentioned is the

default Branch so when you create a repository you're automatically creating a master Branch so this is where your

project will be residing on and now if you want to create another Branch say let's name this Branch one branch

so this is what I want to name my additional Branch I'm simply going to name it and I'm going to click on the

enter button so it will redirect me to a page so this is the exact replica of

your master branch and you can work on this Branch you can work on any other feature or you can add something you can

remove something you can really experiment on this branch and if you're happy with this you can merge back this

feature or the experimentation that you've been working on to your master Branch right so you can see there's a

readme text file it's exactly the same there's the name of your repository the description of the repository you can

click on the readme text file and everything's the absolute same you can quickly switch back to the main page but

the only difference is that you're currently on a branch named Branch one branch you're not on your master Branch

now if you want to switch back to your master branch and work on it you can click on this button and you'll find the

master here you can click on that and it will take you back to your master Branch so you can work on your project so the

currently two branches you can see that and everything's normal everything looks simple that's all for branches it's

really easy I hope it's clear so you can look for branches here on this bar here

that's present here you can also create new ones in the same option so that's

all for branches let's move on to the next part of the session make a comment now what are comets

comments simply record changes to one or more files in your branches so basically

they save the changes that you're making in your project git always assigns each comment a unique identification which is

called sha or a hash that identifies the specific changes so for any changes are

made to your project files you can simply go back and look at the version history or the history of the each

commit you've performed on your project files so that's what really commits are all about it's extremely easy let's go

ahead and make our first comment now I'm going to switch to my Branch one branch and I'm going to make my first comment

I'm going to click on my readme text file that's the only file currently in our repository so we'll make the change

in the readme text file there are three really important icons that are present in the right corner as you can see the

first one is a PC icon that says open this file in GitHub desktop so if you click on this file this entire file will

open in your your GitHub desktop version the next one that is the pencil icon will allow me to edit this particular

file that is my readme text file and the third icon is a bin icon which will allow me to delete this file now what

we'll be working on is the pencil icon that's the edit this file option I'm going to click on this and I can simply

view a space or a file that will allow me to make changes to my readme text

file I'm going to add another line here that says this is my first comment this is what I want to add to my readme text

file and if I want to preview the changes I'm going to click on this preview changes button you can see that

this is my first repository this is my first Commit This is my first commit is the additional piece of information that

will be adding to my readme text file and it's highlighted in blue so we know that that's the additional information

I'm happy with this change I'm going to switch back to my edit file I'm going to scroll down and if I want to add in

description about the change that I'm performing to my file I can do that a

good programmer would always add a description to the change that he's making to the project file so other

collaborators when they view the commit or they view the change they can read the extended description and understand

what the change is about so that's a good habit that you must follow but as of now we're not going to do that so I'm

going to leave this blank as you can see there are two radio buttons that are currently available the first one says

commit directly to the branch one branch this will allow me to make the change or save or make the comment directly to my

Branch one branch only so the change that I'm making currently is only implemented to my Branch one branch the

second option allows me to create a new Branch for this particular commit and start a pull request we're not going to

talk much about this option right now but the first option is extremely important so we let this be stuck on to

the option that says commit directly to the branch one branch and I'm going to click on the commit changes options and

this will simply implement the entire change to the file you can see that the change is implemented the additional

piece of line that says this is my first comment is added to my readme text file

now the interesting part is if I switch to my master Branch the change is not

implemented in my master Branch so the change is only currently present in my Branch one branch and now if I want to

view the history of the changes that I've made I've already mentioned in the previous part of the session that the

history button will allow me to do so so I'm going to click on this history button and you can see that I made my

first comment 23 minutes ago and I made my new comment 41 seconds ago and there

is also a hash number unique hash identification number that allows me to distinguish between both of these

changes so all of my comments that I'll be making on this Branch will be available here so that's the main point

of a version control system isn't it understanding and keeping a record of all the changes that we're performing in

our files and our projects so this is gives full Justice to the word Version Control so that is what GitHub is all

about now that we learn how to make our first comment two let's move on to the next part of the session

open and merge pull requests so what are pull requests pull requests let you tell

other developers about changes you've pushed to branch in a repository on GitHub so once a pull request is open

you can acknowledge and review the changes with collaborators and add follow-up commits after which your

changes are merged into the base Branch so there are two ways to create a pull request the first one being pulled

requests from a forked repository and the second one being pulled request from a branch within a repository currently

in this demo we will work on the second one that is pull request from a branch within an repository now I'm simply

going to switch to My Demo part okay now currently I am on my master Branch I'm going to click on this pull request

option that's here now it says Branch one has had recent pushes three minutes

ago compare and pull requests I'm not going to click on that I'm going to Simply click on the new pull request

option here so this will allow me to compare the changes there's a base branch and a compare Branch the base

launch is the master branch and the compare Branch I will compare my master Branch to my Branch one branch so this

notification says that the merge between the branch one branch and master branch is definitely possible so it's a green

signal so if I scroll down I can view the difference between both the branches so the left hand side indicates the

information that is present on the master branch and the right hand indicates the information that is

present in my Branch one branch plus sign indicates the additional information that is present in my Branch

one branch I'm happy with this so I'm simply going to scroll up and create the pull request and click on that I can

also leave a comment and I can preview the change that's not necessary for now so I'm going to go ahead and create the

pull request now this will redirect me to a page so this page says that the pull request has

been opened and now I can choose to merge this pull request that is I can

merge the branch one branch to my master Branch so it says this branch has no

conflicts with the base Branch merging can be performed automatically and that's good news right so I'm going to

click on merge pull request and I'm going to confirm my merge update my readme text file I'm happy with that so

I'm just going to confirm it now it says pull request successfully merged and

closed you're all set the branch one branch can be safely deleted I'm not going to delete the branch I'm going to

compare both of the branches and see if my master branch is exactly the same as

my Branch one branch so I'm going to click on this edureka I'm going to go to my main page of my repository so my

master branch has the additional piece of information that says this is my first comment now if I switch to my

Branch one branch it has the exact piece of information so the information that was present in

my Branch one branch has been successfully implemented to my master Branch so that's all for the pull

request part two we've reached the end of the demonstration part now let's quickly look at the case study of how

Microsoft implemented GitHub I'm pretty sure most of you have heard of Microsoft Microsoft cooperation is an American

multinational technology company it develops manufactures licenses supports

and sells different computer software consumer electronics personal computers and other related services

so initially Microsoft was against the use of the open source because they held very tightly to the Internet Protocol

they were completely hesitant to adapt to this new concept of sharing code to the entire world but in 2010 they

rethought this entire scenario and now Microsoft is one of the biggest contributors to open source today about

2000 to 25 000 Microsoft Engineers maintain typescript.net Windows terminal

dark Helm and more than a thousand other open source projects so first what they

did was they released new processes in measured containment but later on they released only license software so here

developers can learn from the company's source code but they couldn't really build on it eventually the stigma died

and now even close code like dotnet is open source under an MIT license teams

realize that they need to accept contributions to get feedback and learn from other Developers

to organize and understand this approach Microsoft created their open source programs office which enables

distribution and centralization of knowledge so the ospo provides the

resources and maintainers to manage thousands of repositories and contributors effectively on GitHub even

though Microsoft invests in its tools they expect other individuals and organizations to lead the way Microsoft

believes that github's value isn't in any one feature but its entire Community

GitHub is the place to collaborate it's where everyone is and where most of the

entire world's open source is already happening it's not just a feature but the whole thing

[Music] how do we configure remote repository on

a local machine what we're trying to do is that we have a existing project on our local machine

and we have not added the git remote so what we're going to do is we're adding the get remote

so if I go back to get LS if I do get remote then there is none so what I'm

gonna do is I'm gonna copy the remote URL from here and I'll go to this

project hello world I'll copy the remote URL and I have to

execute this command get remote add and then the remote name and then the

repository name so I'm going to do is origin and this is my repository URL so

if I now do get removed I should see now origin I can also see it verbose let's see so remote name is the name for the

remote and the remote URL is the remote repositories URL that I just copied from the GitHub I can see the detail link

between the remote repository name and the URL which is here so I have my

origin and this is my remote URL for push it's also the same this is another

demo we are doing wherein we are configuring existing project as a remote repository so what we are going to do is

we are going to create a new repository on GitHub we have open the get bash and initialize the project and whilst that

is initialized I'll add all all files in the local repository by adding git add dot then this stage file for the first

comment so once that is there I'll make my first commit as get commit minus M

and then we will add the remote so let's I'll start with this so let's create a new repository on the GitHub

I'll go to new depository

this is public and I'll initialize with 3d.md and I'll take the licenses chain

general public so I'm creating my repository my repository is created now

I can go back here and let's me so I'll create new project my project

too I will add a new file desktop txt and I'll initialize my die screen so get

I've initialized the directory I can look at the guest status now so I have untracked file I'm gonna add it

this is my first comment so I'm committing this file from staging area to the repository now this is committed

now what I have to do is I'll add my ADD remote repository so if I go to get remote to look at the repositories then

they should be numb so what I'm gonna do is I'm gonna copy this remote URL

repository URL I'm gonna do get remote at the name and then the get now if I do

remote minus V I have my get ready now get push origin master

so what we did is we didn't integrated the remote changes so before this the push I'll have to do a pull here because

we have readme.md file so I have my now I

so what I did is there was some issue with my brand so what I did is I just replaced my prawns and if I look here

now I'll do is I'll push it to the origin master

so now I have uploaded the project so what we did is we created a new project

and when we uploaded this file we added the remote URL remote repository URL and we have uploaded the file so we can see

the remote repository that we already knew that we can type get remote and do minus V and it will show us the URLs

that git has stored it will list all the remote servers which are there so this

is what we are talking about git remote I can get origin and if I add minus v i can get all the URLs I can also rename

my remote repository for example instead of origin I want to for example if I

have different remote repositories then I can for example name it something else let's so what I'm going to do is I'm

gonna I'm gonna rename the origin to lapper

one if I do now get remote so I have wrapperman now so this is just for the

renaming a remote repository name now to remove a remote repository we can

use either the below options so they both the same commands first one is get

removed remove and the second one is gate remove RM these two commands the reason for removing can be that maybe

the server has moved and we are not using that particular mirror or our contributor is not contributing any more

in the project so there can be n number of reasons for removing the branch in case the server has been moved then we

first remove the remote repository and then we will add the remote repository

again that we saw in the previous examples the point here to note is that all remote tracking branches and

configuration settings will also be deleted for that particular remote repository so how can we check the remote repository so I'm gonna again do

get remote minus V so I have the gate repository named origin with the git URL

is listed so to remove it what I can do is I can either use get remote RM and

then the branch name now if I check this again then this will return me a blank

result so now let's take an example what I'm going to do is I'll quickly remove my

module 3. so I'll quickly clone this this is the same we are using since I've removed the remote instead of adding the

remote cloning is much faster for me now so I'm gonna get into this and this is I

have the remote available so what I'm gonna do is let's check the branches and then execute the command so we have only

one branch which is Master which is the default Branch so now I'm going to create another Branch let's name it as

Old Branch well now since we will cover this in more detail in the coming slides

but for now you have to understand that when I execute the code get Branch Old Branch or the branch name then this

branches created on the local module then this branch is created on my local

repository and I have to push it to the remote repository so the command to push it is get push then the remote

repository name and the branch name so this branch is moved I'm gonna open

another window for Chrome and now what I'm trying to do is we're going to go to the GitHub and see if the

branch has been created successfully or not so I'll go to my repositories module

T3 and yes there is an Old Branch right now since we have the old branch of

living on get remote so now let's remove this Branch or delete this branch and the command to delete is same the only

thing that I'll add is I'll add dash dash delete after the repository name so

get push origin dash dash delete Port branch so let's go back again to this and see

EFT so now I have only one branch which is master let's create another new branch and

let's sorry not the push but I'll have to first create it so I'll do great Branch New Branch and then get Branch

this will show the new Branch okay the point here to note is that as I said the

when I deleted the branch from the remote and it was only deleted from the remote if you can see here there's only

one branch and this Old Branch is not deleted from the repository to delete this Branch completely or any branch

completely the steps will be first to delete it from the remote second to delete it from the local so to delete it

from the local I'm gonna do Branch minus t and the Old Branch name now if I do

grid Branch I only have one branch which is get and now I'm gonna push this new Branch again to origin and then the knee

Branch name

yes so now I have another branch which is New Branch now I can also go ahead and rename a

branch so to rename branch on the remote repository you need to First rename a

branch on the local repository once that rename is done you will then push it to

remote by with the same name for commands so let's check this for example

here what we're doing is we are checking out a new Branch we are creating a new Branch Old Branch and at the same time

we are also switching that Branch again so this whole one single step can be broken down into two number one we

create a branch with grid Branch Old Branch and number two we check out we switch to the Old Branch using get

checkout Old Branch but this single command will do the both thing it will create a new branch and then switch it

on it so if I do get branch then I have only New Branch so let's do git checkout

so you see it has created new branch and switched to it but this is the local brand this Branch does not exist on the

remote so far there's only New Branch there are two branches master and New

Branch next what I'll do is I'll push this Branch to the remote

so the command is same git push the remote repository name and the branch

name okay now I have three branches Master new branch and the Old Branch

and if I do this command the four I rename so let's rename it to here what

is happening is that when we create this Branch we had only one two branches master and Old Branch and then they are

renaming this Old Branch to New Branch so let's rename this Old Branch to actually New Branch Norm New Branch one

so the command is get Branch minus M the Old Branch name and then New Branch one

if I do now get Branch then I have again three batches Master new branch and the

new Branch one but if I go to the remote again then I have not pushed this change to the remote yet so I have only three

branches Master new branch and Old Branch this has not been renamed on the remote so to rename it on the remote

what I'll do is I'll push this Branch get push and what I'm going to do is get

push call on

if we now go to this branch and let's check the branch name so the Old Branch

name has now been changed to new Branch 1 on the remote as well while if we go

to this example then what is happening is that we have created a new Branch new Temp and then switched to that branch

so the minus M option here is for moving or renaming a branch so generally the

command follows this get Branch minus M the Old Branch name and the new Branch name if if we go and try this example

then get Branch minus M let's call it temp is new Temp and then we are going

to move temp to Newton but the issue here is that they already have temp so let's create first of all temp here okay

if I do now get Branch then I have temp so the First Command what happened is if

you remember our new Branch 1 has been renamed to Temp so if I'm not providing

the second argument it will consider the default branch which is selected or the current branch which I am on as the

branch which has to be renamed now if I'll move temp to new temp if I do now

the gate Branch then temp has been renamed to newtamp in the last step I am

pushing this but let's see if it will push or not but I think it will I'm not

sure if it'll work because we don't have the temp on the remote let's see

it has failed to push why because I don't have the new Branch I have not

pushed the temp so let's rename this to Temp again and then let's put new Branch

one to temp

so now I have the command temp now also

sorry the branch name temp and now if I move this temp to new temp and then push

it to the remote repository then this will work as well

in the same way we can delete the remote Branch we already saw this example but

let's now delete this branch that we created so the command is this push then

we pass the dash delete option and then the remote repository name and then the branch name so minus t origin new

underscore temp login as for password if you are working on remote I strongly

suggest that you should consider setting up the passwordless SSH so this branch

is now deleted let's verify it yes now we have two branches master and New

Branch we deleted the new temp now this is the another option we have in when we

are pushing our changes to the remote which is the minus F or the force push flag so get push minus f is a short for

get push hyphen hyphen Force option if I go and we can try up so

this is the option we are talking about Force so this option will force a push when otherwise get would reject your get

push because you might have changed your repository history in pushing your repository so Force push also lets you

override someone else's comment which have been pushed after your last four

for example let's consider that your repository is at commit a and you change something locally and committed so this

now becomes the commit P you push commit B to your remote repo so both are now B

now now you decide to change something again and replace your comment B by

commit B hyphen now you want to push your commit B hyphen changes to the

remote repository a plane get push will not work so you can now remove commit B

from your remote repository and push B hyphen instead with get push dash dash Force so what will happen is it will

overwrite the changes and will replace your branch directly be aware that Force pushing is a highly dangerous and

unclean solution when you are working in a shared repository so instead of you someone else might have pushed for

example B hyphen and if you're trying to commit now B hyphen hyphen then you'll get a merge conflict and in that case to

resolve the Munch conflict you can override your changes however that is not recommended and that is unclean so

all comments in the shared repository should be considered immutable this is the best practice and to keep your

repository consistent and keep your co-workers happy you should use git revert instead or you should go for

manually merging the conflicts which we'll see in the next module so let's try this git push example let's see how

the get push will work so what I'm gonna do is I'm gonna go to test.txt and let's

update some content in it it says add it from remote let's change it to remote one another line from GitHub let's

commit those changes so what has happening now I have a different version on the remote which we can consider that

someone else pushed it and we have a different version on the local so now let's update our local repository please

note that we are not merging the changes we are only updating our local Repository

I'm assuming we were on Master yes and get branch let's check out the master as well so

now I have also fetched the master repository here now you can see that my

changes are there and but they are not been pushed into my file yet or merged

into my file yet I still have added from remote but not that so if I go ahead and

change my repository for example let's change the test.txt before that first

check the status so there's nothing to commit via clean so let's edit test.txt

and let's add this line This is added from local repository I'm again again

saying I'm using emacs editor here if you are comfortable with Vim or Nano you

can just go ahead and use that now what I did is I updated the file but that is not in the staging area so let's move it

to the staging area now it is in stages and now I am gonna commit it added line

from local rapport a workspace I can say so now we have our remote is on

different commit and my local is on different command so what I'm gonna do is now if I go ahead and if I try get

push origin Master then this will be rejected so let's okay it's

authentication fail let me try it again

so this is rejected why because the tip of my current branch is behind its

remote counterpart so we have to integrate the remote changes before pushing again so now I have two options

either I merge then manually which is called as the merge conflict which we'll see in the coming slides but for now we

can have unclean solution which is overwriting all the changes so what I'm gonna do is I'm gonna just add a force

option here before the get so not before not after the get but after the get push

so you see now my changes have been pushed here and this changes another

line for GitHub has been removed completely so it has overwritten my file if I go to the history then I can see

that I have this added line from the workspace but I don't have that change

wherein I added line from the local from the GitHub once you have merged the force update or

once we have did the force push to the depository let's look at the file locations in both pre and post command

so any file will can exist in three locations which we already know what this location are and the same file can

have different content and different locations so the first one is remote inside the repository so this is the

remote which is for example is in GitHub second is stage in the index so this one is ready for commit and is the staged

area third is the workspace which is not in the staged area but it is in a workspace so far so content of these

file as it was last committed to the repository this is in the repository this is on our local but second one is

in this stage and ready to commit well the first one is not a stage yet and

most files are in progress State get wound or do an auto update for a stage

file with additional edit to do that we need to explicitly we add our files from

the workspace to the staged area using the get add command which we already saw in the previous examples

the next thing we have is compare remote wrappers so we already saw how the

conflicts can happen so this comparing the remote repository will come in handy with the conflicts when we'll come to

that but we can also compare our file with the remote one or between the two locations or between the two files or

commit hash different options if we do not specify any option then it will

compare to the staged or the index version to compare the stage version with the committed version so committed

version is what is in the remote area we have to use dash dash test additional different option are comparing file

within two files on the desk comparing the two comments or comparing file without any comment so the command is

get diff which is this let's look at the help for get div so this will show

changes between comets commit and the button tree Etc we have different options options can be dash dash cached

or and we can compare different files so let's do a specific first thing so let's

compare we are not providing any option the workspace to the staged version so let's do get div test DOT txt first edit

our test.txt if I look at the status now we have up to date right so let's edit

test.txt and let's add this this line is forget F now let's compare this file so

if you see we are comparing it with the index version we have a b and we have

added this line this line is forget diff this will appear in the green color but if I go ahead and remove the line

and if I do get diff then what is happening is I have a red color which is seeing this line has been deleted and I

have a plus color there is a blank line inserted and then there is another line inserted now what I'm gonna do is I'm

gonna add this to staged area if I look at the status now then this is added the

stage area now if I do get div then there are no changes because I'm comparing test.txt to 1 that is stage to

compare it with the remote I have to use cached this is added from the remote the

same comparison is the same that this line is deleted and these two lines of the one blank line and one additional

line has been added but this is now compared to the remote I can also go ahead and compare the two files so let's

create another file a test dot text oh let's rename it to New test

Dot txt and let's add this is added from remote sorry not remote let's make it

from local now I can use git diff to compare these two files not get there if

I can directly use the dev command test.txt and new test.txt so I can

clearly see that I have these changes this is my line one this is my file one

which is test.txt and this is my file 2 which is newtest.txt and the changes

have been are differentiated from this line dash dash if I copy test.txt now to

newtest.txt then these are integral but if I go ahead and just added a single

word then you see the only this line is coming as the difference so there is f

missing but this complete line will be treated as a different line we can also go ahead and compare the two commets so

we check this options We compare two files now we can compare two commits as well and how to do that commits can be

identified with the commit ID so if I do git log I'll have the commit ID now I can compare them so let's do get diff

let's copy these so I'm gonna copy two different commit IDs and if I do this

then it is saying me that there were two files test.txtntest.txt and this line has been

removed if I look at the their comments then this is 965 which is update text file

let's compare these two let's compare the latest one and this

one so this line is not there in the first file which is test.txt which is the

latest comment this is added from local repository hats is there is not there in

the other file we can also go ahead and compare file with any comment so how can

I do it you can again let's look at the log and then compare with let's copy

this what I did is I compared this commit ID

with the workspace file now there are other options as well that

we can compare with any two files on the disk which we already saw we can compare

two commands we can compare any file with any comment now to compare local branch with the remote we already saw

how this can happen that someone else might have contributed in the in his copy and then push that changes to the

remote while we have not pulled the changes yet so first what we have to do we have to

update our remote tracking branches how do we do that we do get fetch and the remote Branch name now remote Branch

name is origin for example which is the default one yes what I can do is I can

do git diff Master branch and the remote Branch path what is the remote Branch path how does it different from the

master so if I do get branch Master is my master Branch path while

remote Branch path will become origin slash Master the remote Branch name slash the remote repository name slash

the branch name let's go back to this again let's open some file let's edit it

and then it will get fetch so what I'm gonna do is I'm going to add compare remote let's remove this

let's commit this file for compare now

what I'm going to do is I'm going to fetch this changes to the repository get fetch origin repository name so what is

the other we update the remote tracking branches so this is my remote tracking branch and this is my master Branch next

what I'll do is I'll do a quick def get div master and then origin master

what we saw is we have compared the master Branch with the remote Master Branch right origin slash master so what

we did is we first fetched the remote tracking branch and then we compared them

now this command will list all the branches so you can see that Master knee Branch

new tab and then wheat is also showing the remote branches so we can choose the branch name from there for example this

option that we already saw Masters origin slash Master we Masters the local Master branch and origin slash Master is

a remote name lay origin and master Branch so origin is the remote repository name and master is the branch

name so folk is the another command that we have in the gift Club so what happens is

that we will it allows us to copy other repositories for our own use so we can

experiment with the changes or we can go ahead and start the development from that Branch so this is like a very

popular and the useful feature of GitHub mainly so it acts as the bridge between

original repository and the personal copy by personal copy we mean that we can copy the data the complete meta

return the repository and then we can start working on it the only thing we won't get are the settings so this is

core of the social coding which is called as open source at GitHub so folks are used to suggest changes changes or

to use someone else's project as a beginning point for your own for example if someone has done this is very popular

when you come to blogging for example if you're using some theme or some blogs or maybe you're building some analytics

project of your own and someone has already contributed to the GitHub then you can focus that repository and then

you can start contributing in it of course you will have to check the license if you're building that product for or developing that code for the

remote for the production use then you will have to check the license and then

you can start contributing so the best example of using folks is for bug fixes

where you poke the repository for fixed issues and raise pull requests to merge in with the original Branch so it is a

two-step process one on GitHub we go to the repository and click on the four button after folk to work on local

repository you can either clone it or download it for example so what I'm going to do is I'm gonna go to some

username for example and I'm gonna go to let's say the bit

start not the best starter I'm gonna go to the straight cardinalizer and I'm gonna Fork this repository for my use so

if I fork it so now this repository is available for

my use right I can now use this as a starting point from a project and to use

this I have to clone it again so if you see the repository name has my username

the repository URL has my username so I go ahead and I clone it

so I have this repository at my this local workspace for my use so I can add

anything and then come and push it for example let's add this file this line

should be treated as dummy I add this and I come with this added Dummy Line

and now I push it so let me check the first Branch get and the remote get push

origin and then Master then it will ask for my credentials

so if I go here and check the history then I have this Dummy Line available

but if I go there for the place where it is forked from then you can clearly see

that the change does not exist there so I have this copy of the project as my

Repository git is a largely distributed which means there is no inherent concept of upstream

and downstream it depends on the way data flows for example the other wrapper will be Upstream even well data is

pushed onto it and if I am pulling from the other repo then upper other repository is Upstream in my repository

is Downstream for example the incredit analyzer project this the repository of

this user is Upstream while my copy is Downstream this is relative between the

two repositories so two sync changes which are in the original repository which we have four we configure a remote

that points to the Upstream repository and to do it we have to do what we have to just use this command get remote add

upstream and the repository name so if I do get remote minus V let me first copy

the repository URL and if I go here and if I do then I have to add this I have added the

Upstream if I didn't do the remote minus V then you can see I can see the Upstream URLs so both Fetch and the

remote which contains the original owner of the Repository I can also do this

thing so instead I can use the Upstream repository name and then I can check out

the folks local branch so this is the Four Queens the downstream repository so

this is similar to other this will work like at any other repository on my local if I do get Branch then I can see

there's only one branch but the remote are different we can also consider merging it so

instead of using origin I can use Upstream

we already up to date I can also do the get diff or let's say not a diff but the

difference is that this line should be treated as dummy does not exist in the Upstream it is only in my remote

repository so this is the change that I can see why it is only showing the

branches Upstream so I cannot go ahead and make changes in the Upstream because

I don't have the ownership I can suggest changes and then it will be up to the approval of it however if something has

been changed in the Upstream then what I can do is I can merge it with my local repository which will be similar to pull

so it will pull the data onto my local repository and I can work on it so now

coming to pull pull is the very useful command we have been using it so this is like opposite of portion push we are

pushing from local repository to the remote while in pull we are downloading the data from remote to the local so

that downloaded data has been pushed by some other user for some other comment or from from the feature might be they

have added some feature or they have added some bug fix or they have changed something so we are downloading we are

updating our repository on our local so this is the source repository and we

have feature source as the branch name we do a pull request and then this is our destination repository and this is

the master destination repository this is the branch name the master Branch name so how it will work is that if

someone has added something in the destination repository then I can pull it on to my local Repository

so this is commonly used by this is a very general command in the get which is

commonly used by teams and organization which have more than one contributor so developers work on their feature branch

and raise a pull request to merge to Shared Repository what it means is that that we can have multiple branches and

then we can have a pull request created open source projects usually pull request to manage changes from different

contributors because open source projects have many contributors who are contributing to them so they use pull

requests to manage the changes now how do we create a pull request now after we

have made changes to the fork after we have forked the project we create a pull request to ask Upstream request

repository to merge our changes we are not committing them directly the word here to note is the ask so below the

steps we confirm the repository we want to change we select the branch of the Upstream repository to merge and we use

the base Branch from the drop down one menu once we selected the document menu we select your four using the hackful

drop down menu which is this and then we select the branch we want to make changes from compare Branch doctor menu

so let's take this example so let's push we already have for something right so

what I'm going to do is I'm creating to create a pull request before I go here let's read what is saying so pull

requests help you collaborate on code with other people as pull requests are

created they will appear here in searchable and filterable list to get started we should create a pull request

so let's create one this is our base Branch base branch is the original

Branch this is our head folk so the base Branch name is master again we have only

one branch and the master now we can see the comparison we have added one line

and there is no change as such this commits has been made by dudika global

which has added a dummy line these are the comments and this is done on December 3rd 2018 so we have one change

file with two Edition one is the blank line and second is the line we added now you can also see the split view

right this is the split view this is the base copy and this is our working copy the address GitHub so let's create a

pull request

so now we have added a dummy line and it says it is not wash but it says it wants

to merge one commit into aishri Master form advika GitHub Master Branch this

branch has no conflict with the peace punch

now if I go here then we won't be seeing this because till the time someone approves it now this is the pull request

we are seeing before we do that let's see if this has been added in the readme.md which we are seeing here we

won't be seeing here because this has not been approved but we can go to pull request and we can see it here so let's

try to open this as a public user so we're getting the whole history we are

saying that we this is the commit and this is the file changed so we can clearly see once someone approves it

then this change will actually appear in the original repository which is here so

This Is How They manage with open source project which have millions of developers they forget they work on

their own and then they create a pull request so we are providing title and description and then we can also control

the access which is appearing at the bottom which allows for edits for maintenance so if someone else can might

change your pull request make a small change here and there we click on pull request and it will be available

[Music] workflows come in all shapes and sizes

teams developing web applications have slightly different needs than those

working with embedded software and I'm sure you have probably heard more than one recommendation on the best workflow

with experience it is said that the best workflow is the simplest workflow it is

easier to teach and follow both of which result in a more consistent application

before we begin today's session I'm quickly going to break down the agenda firstly we will understand what actually

is a workflow and then we will see some differences between git flow and GitHub

flow moving on we will understand some of the issues we could face with Git flow finally we will address the main

subject of today's session that is the GitHub flow why GitHub workflow is

important now if you like our videos please don't forget to subscribe and press that Bell icon to never miss the

latest updates on the edureka channel also do check out the edureka devops

training certification the link is given in the description box below now let's move on to the first part without any

further Ado what is a workflow workflow is about controlling the sequence of

events that transform a piece of work to ensure every person on the team can use

the best tool at the right time to put it simply a workflow is how you get work

done it is a series of tasks you need to complete in order to reach some repeatable business goals at its core

the workflow is change management as I have already mentioned before workflow

comes in all shapes and sizes there is no Universal solution for every team's

needs an optimal workflow enables the systematic organization of resources

into processes that transform materials provide services or processes

information the principal ways to manage software in git are the git flow and the GitHub flow

these two methods can really help you to manage a project and optimize your entire workflow in the team now that we

know what is a workflow let us discuss about git Flow versus GitHub flow so

first we'll discuss about git flow git flow works with different branches to manage each phase of the software

development it is suggested to be used when your software has the concept of release because it is not the best

decision when you work in a continuous delivery or continuous deployment environment with this particular concept

is missing another good point of this flow is that it fits perfectly when you

work in a team and one or more developers have to collaborate on the same feature

so the main branches in git flow are Master develop features hotfix and the

last one is release now let's discuss about GitHub flow GitHub flow is a

lightweight branching workflow that supports teams and projects where deployments are made regularly it is

centered around a particular feature and small confined changes so now I'll discuss some of the key features of

GitHub here the master branch is always production ready Master is also deployed

to production development often only happens on the feature branches the

deployment speed is extremely fast here and the last important feature is that it requires a verification for each

release now let's move on and discuss some of the major issues of get flow so

here many of the developers do not like that new feature branches are started off the develop Branch rather than the

master Branch or the way it handles hot fixes but these are fairly minor issues

but before we move on with the major ones let me explain you what a hot fixes hot fixed branches are a lot like

release branches and feature branches except they are solely based on Master

instead of the develop this is the only branch that should Fork directly off of

the master now moving on to the major issues with Git flow it is much more complicated than most developers and

development teams actually require it to be it's complicated enough that a big

helper script was developed to help enforce the flow though this is okay the

issue is that it cannot be enforced in a git graphical user interface but only on

the command line so the only people who have to learn the complex workflow really well because because they have to

do all the steps manually are the same people who aren't comfortable with the system enough to use it from the command

line this can eventually be a huge problem these issues can be obviously solved easily just by having a much more

simplified process so here at GitHub they do not use git flow they use a much

simpler git workflow its Simplicity gives it a number of advantages one is

that it is easy for people to understand which means they can pick it up extremely quickly and if they ever mess

up they can always undo another is that we do not need a wrapper script to help

enforce it or follow it so using graphical user interfaces and such are

not a problem so the next question that arises is why don't they use git flow at

GitHub well the main issue is deployment the git flow process is designed largely

around the release Branch GitHub does not really have the release is because your deployment to production occurs

every day that is often several times a day the process of testing and shipping

is as simple as possible so that every employee feels comfortable doing it now

let's move on and understand the main topic of today's session GitHub flow the

first step in GitHub flow is to create a branch so when you're working on a project you're going to have a bunch of

different features or ideas in progress at any given time some of which are ready to go and others which are not

branching exists to help you manage this particular workflow so now when you

create a branch in your project you're creating an environment where you can try out new ideas changes you make on a

branch do not affect the master Branch so you're free to experiment and commit changes safe in the knowledge that your

branch won't be merged until it is ready to be reviewed by someone you already collaborating with you can always switch

branches using the git checkout command now branching is a core Concept in kit and the entire GitHub flow is solely

based upon it there is only one rule anything in the master branch is always

Deployable you should remember this so because of this it is extremely important that your new branch is

created of the master when working on a particular feature or of particular bug fix your branch name should always be

descriptive so others that are working with you can see what you're working on now let's move on to the second step in

GitHub flow add commits so once your brand has been created it is time to

start making changes whenever you add edit or delete a file you're making a

commit and adding them to your branch this process of adding commits keeps track of your progress as you work on a

particular feature branch make sure you are on the correct Branch by listing it and if necessary switching to it by

using the command git branch and then using the next command get check out

your feature Branch name make sure you are on the current Branch by listing it using the get Branch

command and if necessary switching to it by using the checkout command comments

also create a transparent history of your work that others can follow to understand what you've done and why each

comment has an associate comment message which is a description explaining why a particular change was made furthermore

each comment is considered a separate unit of change this lets you roll back changes if a bug is found or if you

decide to head in a different direction you can do this by adding committing and then finally pushing it comment messages

are extremely important especially since git tracks your changes and then

displays them as commits once they finally push to the server by writing clear commit messages you can make it

easier for other people to follow along and provide feedback next step is open a

pull request pull requests in initiate discussion about your comments because

they are tightly integrated with the underlying git repository anyone can see exactly what changes would be merged if

they accept your request you can open a pull request at any point during the development process when you have little

or no code but want to share some screenshots or general ideas or when you're stuck and need some help or

advice or even when you're ready for someone to review your work by using github's at mention system in your pull

request message you can simply ask for feedback from specific people or teams

so in pull request you can also describe how to test your changes pull requests

frame a problem by describing it maybe you're trying to fix a bug add a feature or trying out a new code pattern having

a proper problem description and shows that your reviewer knows exactly what you're getting into to let your reviewer

know how to run your code describe how to test it properly lay out a scenario that gives a basic run through from

setup to the expected outcome are there any multiple angles you can test your feature with awesome specify a test case

for each scenario not only does this help your reviewer navigate through your code it also helps you by describing the

feature step by step for someone else it ensures you have not missed anything critical now let's move on to the next

step that is discuss and review your code once a pull request has been

finally open the personal team reviewing your changes may have questions or comments you can obviously read these

you can also continue to push to your branch in the light of discussion and feedback about your comments if someone

comments that you forgot to do something or if there is a bug in the code you can immediately fix it in your branch and

push up the change GitHub will show your new comments and any additional feedback you may receive in the unified pull

request view pull request comments are written in markdown so you can embed images and Emoji use pre-formatted text

blocks and other lightweight formatting you can even suggest committable changes

right in the comment a review basically focuses on the code health and on

production readiness so let me explain you what code health is perhaps the coding style does not match project

guidelines the change is missing unit test or maybe everything looks great and props are in order pull requests are

designed to encourage and capture this type of conversation you can suggest changes directly in a comment by hitting

the suggest changes option now let me explain you what production Readiness is the pull request template lays out one

or more scenarios on how to test your code the reviewer will go through a scenario and document the results and

whether or not the test is passing the reviewer should be generous with comments descriptions and screenshots

also ask for any sort of explanation or improvements as the test is progressing

the reviewer will also describe if the test scenario passes with this we'll

move on to the next step that is test your changes so before any code is finally deployed

into production it needs to be test in a production-like environment that is you could say the staging area to make your

life easier read upon how to make an update script to update your tool for each environment right here with this we

come to the last step of GitHub flow that is merge your feature now that your changes have been verified in a

production-like environment it is time to merge your code make sure the code has been signed off before proceeding

squashing commits offer pull requests improves the readability of the commit history so during development commit

early and often once your work is ready merge with the focus commit message by

default the merge button for repositories will be set to squash and merge once merged pull request preserve

a record of the historical changes that is made to your code because they're searchable they let anyone go back in

time to understand why and how a decision was made it I would like to conclude this session by reminding you

how GitHub is an amazing tool to take advantage of you can create repositories

and issues create branches Fork projects make comments submit pull requests for

reviews and merge to the master Branch so I would suggest you to work collaboratively on projects using GitHub

thank you so what is branching nearly every VCS

has some form of planting support branching means you diverge from the mean line of development and contribute

to do work without messing with that Main Line in many VCS tool this is somewhat expensive process often

requiring you to create a new copy of your source code directory which can take a long time for large projects some

people refer to Great's branching model as its killer feature and it certainly

sets get a part in the VCS Community the way you get branches is incredibly lightweight making branching operations

nearly instances and switching back and forth between branches generally just as

fast unlike many other VCS get encourages workflows that branch and

merge often even multiple times in a day understanding and mastering these features gives you a powerful and unique

tool and I can entirely change the way that you develop so the mean another main difference is is that that

developers can create their own branches and can work in parallel this keeps the main Master Branch free from the

questionable course for example if this one is the master Branch then you can have two more branches before rocking

and power one can be the feature and other can be the bug fix one of the biggest advantages of get is its

branching capabilities unlike centralized Version Control System get branches are cheap and easy to merge

this facilitates the feature Branch workflow Which is popular with many git user the feature branches provide an

isolated environment for every change to your code base and when a developer wants to start working on something no

matter how big or small they create a new Branch this ensures that the past Branch always contains production

quality code using feature branches as not only more reliable than directly editing production code but it also

provides organizational benefits so in this example we have a master branch and

then in between we created a test branch and later on created a feature branch because branches are lightweight and

they are very fast switching back and forth so git branch is essentially an

independent line of development you can take advantage of branching when working on new features or bug fixes because it

isolates your work from that of other team members branches serve as an abstraction for the edit stage or commit

process you can think of them as a way to request a brand new working directory staging area and project history when

you want to add a new feature of fixable no matter how big or is how small you

spawn a new Branch to encapsulate your changes this makes it harder for unstable code to get merged into the

main code base and gives you the chain chance to clean up your features history

before merging it into Mains Branch their fault Branch name is master this

is created by getting it Commander new repository we will pointer to a snapshot of your changes and it moves with the

changes you commit so these are the branches operations we can create a new Branch or we can create

and switch to the new Branch directly to create a new Branch we have to use grid branch and the branch name and to create

and switch the new Branch with a single command we can use get checkout minus B and the new Branch name so let's try

this if I'm in module 3 and if I check get Branch then I have three branches

New Branch new Temp and temp if I create another Branch bug fix now let's create

test let's stick to fuckfix then I'm still on Master Branch to use the bug

fix I have to use choose get checkout bug fix now there is a simple way to

create and switch to the branch at the same time and the command is

so now what happened is we replace two commands with the one so command get

branch name and the second command get check

out Branch name is replaced with get checkout minus B Branch name

to list branches for a particular repository or to locate repository and

click on the branches as highlighted so we already saw this but let's go back

there again let's choose and depository let's take the most dual 3 only we click

on the branch here and then you can see all the branches we have two branches remove branches master and nebron we can

also check that using minus a option so we have two remote branches master and

new branch and four local branches let's discuss the gets storage strategy so get

does not store data as a series of chain sets or differences but instead as a

series of snapshot when you make a comment get stores a commit object that contains pointer to the snapshot of the

content use stage this object also contains the author's name and email address the message that you type and

pointers to the comment are commits that directly came from the comment it's parent opinion zero parents for the

initial comment one parent for normal combat and multiple favorite parents for combat that results from a merge of two

or more branches to visualize this let's assume that you have a directory containing three files and you stage

them all and comment staging the files usually compute a checksum for each one in toasted version of file in the git

repository and adds that checksum to the staging area so after staging two things

are happening get computes this action and install each version of each file and add to the staging area

when you create the comment by running the get commit get checksum each subdirectory in this case just the root

project directory and store those three objects in the git repository git then creates a commit object that has the

metadata and a pointer to the root trajectory so it can recreate the snapshot when needed now get places only

four types of objects in the object store the blobs trees commit syntax we

already discussed them in the past module now these four Atomic objects form the foundation of gets higher level

data structures first the blobs the each version of file is represented as a blob

blob which is a contradiction of binary large object is a term that is commonly used in Computing to refer to some

variable or file that can contain any data and whose internal structure is ignored by the program for each file

there is one blob which stores the sha1 hash of the file content in it next come

the trees we which represents one level of trajectory information that is a

hierarchical structure which contains information about each of the blob it creates relationship between directories

and the files in it next is the commit the commit objects holds the metadata for each change introduced into the

repository including the author committer commit date and the lock message each commit points to a tree

object with all metadata next to the tags so we have not discussed tagging so

far but tagging is generally used to capture a point in history that is used for marked version release such as

v1.0.1 a tag is like a branch that does not change unlike branches tags after

being created have no further history of comets now get suppose two type of tags

annotated and the lightweight tags which differ in the amount of accompanying method data they store a best practice

is to consider annotated Taxes public and lightweight taxes private annotated

tags to extra metadata such as Dragon name email and date this is important

data for a public release lightweight tags are essentially bookmarks to a commit there are just a name and a

pointer to a commit which is useful for creating quick links to limit relevant comments

now this is the storage strategy for object structure now if you see we have at the first level we have blocks where

each file is represented as a blob next We have tree which contains content of directories and specifies files stored

as blob so we have a tree which tree name we have a blob the hash and the file name and when we have comment which

is a pointer containing the metadata and pointer to root 3 so the this is the commit ID then we have tree which is

Reid then we have author Steve coming to Steve in the parent jb007 now our creating a new comment the

new commit will store a pointer to the commit preceding for example this so this is jb007 and the parent of this

commit is this now the parent of this commit is this so this is Snapshot a b

and c now how get internally manage these branches let's say you created a

new Branch Testing using below command get Branch Testing this will create a

new pointer to previous command the previous commit as this 0bi Y2 now git

also keeps an additional pointer which is called as head so head will always point to the last commit of the branch

for example if the branch we are on Branch Master then head will point to this now if we check out a new Branch

Testing then head will move to the last commit of the test branch and when we make additional comets in

the test Branch the head will move forward or the pointer will move forward so there are two types of branches local

branch and a remote Branch local branch is a branch that only you that is the local user can see it exists only on

your local machine and the command to create is get Branch at your branch name while the remote branch is a branch on a

remote location in most cases or the default remote name is origin you can

push the newly created local branch to origin now other users can track it and

can see it so the default name from local branch that gets created when you

run get in it is the known as Master which is traditionally it is also known as trunk so let's quickly do a demo here

let's go to Temp let's remove temp and let's go to tem oh mkdr10 I'm gonna do

now now getting it into it now wait let me first remove it

now I'm going to create a file into it touch file.txt now let's initialize get

Branch coming there's some issue with my configuration ideally it gives you the

branch but if I do get status I should get I'm on Branch master so this is the default Branch I'm getting now I can

create additional branches such as def feature and Bug these all constitute of

branches created for the developer so that they can work parallely and independently on individual tasks

without any inference now certain conventions are provided as best practices for example feature Branch can

be named as feature bug branches bulk release branches release a remote branch

on the other hand is local copy or the remote tracking Branch I should say is a local copy of a remote Branch when a

branch is pushed to origin using the command a remote tracking Branch name

origin slash Branch name is created on a machine this remote tracking Branch tracks the remote Branch you can update

this remote tracking Branch to be in sync with the remote Branch using git fetch or get pull now let me show you

what does the remote tracking branch mean if I go to module 3 so now this

origin slash Master is my remote Branch I don't have a remote feature Branch

feature is only a local branch if you see here this is just the local branch however this is a remote branch which

has a remote tracking Branch origin master now let's see the branches example so

when a git clone command is executed how do we execute the git clone we just get the copy the remote URL we got the

remote repository URL and then we perform the execute the command pet clone now when we do git clone then git

will pull all data from the remote repository to my local which is remote repository to the local now git

automatically names the Clone repository to origin this is the default and the pointer is created at the master branch

which is known as origin slash Master a local Master which is this is also created at the same position for local

operations now switching between branches and different commets to switch to a particular Branch we already know

how to clone it but before we move any further let's clone it again so I'm gonna remove the module 3. let's need

the module 3 as it is okay I'm going to read the module 3 and I'm gonna clone it

again so I'm gonna move into the module and

let's look at the branch we have one branch and the one remote tracking Branch so to create a branch what I can

do is I can do get checkout buck another checkout sorry to switch between

the branch we have to use git checkout Branch but before that let's create a branch I can use bug now and if I do

great Branch then I am on Branch Park now I have created a bug Branch now

there are few other options I can create a branch so the first one is I can

create local branch from the head and can switch to it automatically so instead of creating get Branch I can use

get checkout minus B let's call it feature so I have created a new branch

and I have switched to it as well so in general get Branch Branch name and

command or I should say yes and command get check out your branch name is

equivalent to get checkout minus B Branch name now another one is we can

create a new Branch from the commit hash so let's look at the commit hash now this is the commit hash so I can use a

new branch and I'll so if I do get Branch I can see that I have a commit hash if I do get

log then I can see that that this branch is create the commutation is Created from this commit

ID I can also create a branch from four for example for forehead prior to commit so

the command is same so if I do get checkout minus B my Branch name add

prior and then I do head four so this will be like this is the

head then it will go like first of all let me go to master and if I do get log

then I can see that I was unmaster only so let's create another minus B new

Branch let's call it head prior and head root 4. so ideally if I do root 4 then

this is like add one two three four this should be somewhere around here

so it's switched so get log now if I do gets logged then this is the

fourth one two three four and this one as I said these are all different ways of creating a branch and switching

between them so to switch to a particular commit we need to use commit

hash and commit hash as this for example if I want to switch to this

one I can use git checkout and then the hash but when we do this our head will

go in a detached site so we can look around experiment and we can create a

new Branch using minus P for example give checkout minus B and dash if I do

get Branch now you can see this is there I can do git log and first check out

master and if I do now get log I can see that detached is from this commit ID so

commit hash is a generated sh1 hash for a particular comment and can't be

changed until the user changes the comment details now we can also delete a branch this might be coming handy so for

example they might be change scenarios when we spin a new Branch for example feature we merge it back onto the master

so that it goes into production so feature Branch now for example has no use and can be discarded as it is

already merged in the master so the way to do it is use the minus D option or

minus capital D which is equal to force option so if there is unmerged branches we use minus D it's the merge branches

we use small D so let's get Branch first of all see the branches

them and I do get Branch minus D Park if I do get Branch now this is treated now

the point here to note is that this has only deleted the local branch this is

all not deleted the branches into the repository which is the remote Branch so

how do we delete the remote Branch if I do get minus a branch minus a v c both

the branches so we have only one branch so far so let's push one branch on let's

push the bug feature branch on the remote this will ask for my credentials

so now how can I delete this Branch so I can do like to delete a branch I can do

get push minus D the remote name and the branch name here remote name as origin

and the branch name is feature this will again Ask for credentials

I strongly suggest you to use the passwordless SSS setup in the git now

the remote branches kit is been removed but only removed and now if you will see the feature is still accessed because

this is the on the local Rod another way to remove that is using get pushed remote name call on the branch name for

example let's push the feature again we are deleting it we're not pushing it so let's remove the minus D and let's push

the feature again

to remove it I can only do add colon here but before that let's first look at

the branches so I have a feature branch and to delete it I just add cologne

before and it will delete the branch for me

next comes the merge conflicts so for a lot of people merge conflicts

are as scary as accidentally formatting the hard drive a great thing about having get as your version control

system is that it makes merging extremely easy and in most cases get

will figure out how to integrate new changes however there are a handful of situations where you might have to step

in and tell get what to do most notably this is when changing the same file even

in this case git will most likely be able to figure out on its own but if two

people change the same lines at the same file or if one person decided to delete it while other person decided to modify

it get simply cannot know what is correct it will then Mark the file as having a conflict which you will have to

solve before you can continue will work so in this example we have this file and

these both are modifying the same file and the same line the third line so this

is visible Creator merge conflict so much conflicts can occur on merging branches or when merging Force history

together so merge conflicts are resolved manually by users git provides different merged

tools to compare and choose the required changes which we will see in the coming slide all user can see use the third

party merge tool with Git so let's look at the example that this is a file and in the master branch and

we add some line in the master Branch again from the base Branch we create a

new branch called feature one and we add another change in it so what happens is that when we merge these Branch with the

master then we'll get a merge conflict so let's first create this example so

I'm gonna go again and now if I look at the branches

now let me remove the old branches which might confuse us

and the head fryer as well right if I do the grid Branch then I

have feature and master let's also remove the feature branch so get Branch minus D feature now let's

edit the test.txt and let's remove it from base branch

now let's add it to the staging area and commit it now let's create a new Branch feature

one now that's the name is features plane now let's edit the test we are still on

the master Branch so I'm gonna use right from Master branch

now same I'll do again I'll add it and then I'll commit it with command test

Master now let's check out to switch to the feature Branch change the same file

and let's write from feature branch do the same things added stage area and

commit it now get let's go back switch to the master Branch again now we

already see that there is some point of conflict that will arrive but let's merge it the feature Branch to master

now here we get a merged conflict right it says that we have a conflict in test.txt and we have to fix it and then

we only we can commit so there are many ways available but the one quick way

will be that we use some tool so what we're going to do we are introduce some tool which is using kdif3 and the

command is git config we said we all we already discussed get conflicts in the past module as well so if you don't if

you have any issues in them or if you are not sure what they are it's request you to go back to the previous module

and just brush this up so we are setting a global config we are only defining merge tool as kdf3 now once that is done

we just run this command get merge tool and this will open us file and then we

can select which merge we want to see and which not so here what we get is we

get the total number of conflicts one number of automatic to solve complex zero number of Unsolved conflicts one

so we have the base Branch we have the local branch and remote Branch remote branches feature localized Master since

we are on master and the basis the parent branch now we saw the base version this column

as displayed here represent the changes which will be made in this area so we can go to the click on the letters and

then it will enable us to see what we want to keep for example if I click on B if I click on a then no Source line if I

click on B because there is no source source line the master hence we're getting this we are not updated we added

a line if I click on B then I get from Master Branch if I click C then I get both if I remove B then I can keep only

feature so let's keep the both and let's see if this file right now once that is done I can open my

test.txt and look at this I can now commit it again let's look at the status

I can commit it also have a look at the this file

so this is how it will resolve the conflict in case there is no tool we are using so this line has no conflict from

base Pawn chance it is above then we have conflicts in this second line so where what is saying that for the

branch Mass head branch which is pointing to master this is the line from master and then we have the separator

line and then we have the line from the feature branch which you're trying to merge from feature brought so we can do

it manually how we can delete these lines and I can decide that I want to keep them both so I just do keep them

both so let's now discuss about stashing first of all let's look at the

requirements that fire station came into picture so often when you have been working on part of your project things

might be in a massive State and you might want to switch branches for a bit to work on something else the problem is

you don't want to do a commit of half done work just so you can get back to this point later the answer to this

issue is the git stash command the stashing takes the dirty state of your working directory that is your modified

untracked files and stash changes and saves that Honest track of Unfinished

changes that you can reapply at any time please note it does not considers the

untracked files stashing stash the changes in dirty working directory but

dirty we mean the work that is not completed yet and is in progress stashing command only takes the

uncommitted changes it they might be in the staged area or they might be in the

unstaged area but they are the track files so stash command flow is it records the current state of working

directory and index it saves local uncommitted changes separately and reverse the working directory to had

commit after this you will get a clean Branch once that is done you can reapply the stash changes again when you come to

that Branch using stash pop or stash apply command and in case you don't want to apply any stash changes further you

can just clear the stash so these are the changes for stash for stash to local

changes by local we mean local branch we use the git stash command to list the modification stashed away by this

command we can use the get stash list and to inspect or check the details we use K it's Dash get show stash so let's

look at this so we are going to continue using the what you three project so

let's do git status so we have this untrack file and now I'm

going to change the test.txt now I'm gonna remove all this and say

this is for stashing demo now if I do get status I can see that

test.txt is modified at the same time let's also change some of the file let's

change the readme.md this is for stashing I'm gonna put this

readme.md to the staging area now so if I look this status we have one in the

staging area one is an unstaged area and the last one is an untracked file if I

look at the git Branch so I have feature branch and master branch and we are currently on feature if we for example

let's first check out the try to check out the feature Branch there's a typo in

my command so it does not allowing me to do a checkout of feature Branch why because I have changes in the staging

area and it is asking me to commit my changes or stash them before I switch the branch so if I look at the branches

again maybe I'm still on master and now I'm going to do git stash so I can now

go ahead and change the branch and if I look the guest status then I only see

the untracked files while the files which were unstaged area or which were in the staged area are not present

let me revert back to the master branch now let's list the stash changes we have

one stash which is this let's also show the changes in it so here is that we

have a commit ID this is the commit ID and the merge the author name date and

some other commands and now I can see the changes so I have two files one is

the test.txt when I have removed these three files and if I do these status

then let me pop it Dash pop so I have two modified files readme.md now the

point here is that the files which are in the staged area for example readme.md was in staged area they now have been

moved to the unstaged area when I do stash so this is the git status now I

can apply them directly now I have to do git stash first of all let's look at the list no so let's stash this again and

now let's do git stash apply so once I do apply I get the same thing if I do

the guest status now now this is the demo that we just saw now at the same time I can use the get stash clear

command as well so let's look at the different options that we have in the guest Dash so we have less show Drop Pop

apply Branch clear create store we can use these different options for example

list that we currently saw we also looked at the show pop apply I can create a branch as well starting from

the committed with the stash was originally created I can remove all these stash entry using clear I can

remove a single stash entry from the list of stash entry I can create a stash entry and I can store this Dash created

wireless dash create these are some examples as well so you can check them

in your workspace so what we're doing here is if you do the git status we can

see that we have two files in the unstaged area the more readme.md and the

palm.xml after stashing the changes no changes are left in the branch and guest

status command shows that the branch is up to date now what we did is if we do

the list then we already saw this when we did the get show then we see the difference is readme.md and it is

pom.xml same as the case that we saw if we go back here right so we have test

dot txt I don't know why we are not seeing the readme.md maybe that was

because it was in the unstaged area part if I do the git status now and if I

remove this test.txt dot college and if I do get stash now then let's do the git

status so it is up to date if I do git star showstash then now I see the

readme.md it was not displaying the read meter MD because it was in the staged area and moved to the other stage one we

can apply the get stash using get stash apply but if you look at the status so far it is up to date but if I do get

stash apply then we have moved these files in the unstaged area

we can also use the get pop saying that we already saw

the rebase command has a reputation for being a magical git Voodoo that

beginners should stay away from but it can actually make life much easier for a

development team when used with a little Caution so we are going to start out with introducing you all to the concepts

of rebase what is it and why do we actually need it while also discussing a

few differences between git rebase and git merge then we shall dive straight

into some rebase commands to implement and discuss a few configuration options as well

finally we'll talk about a few pitfalls of rebase so what is git rebase

rebasing is the process of moving or combining a sequence of commits to a new

base command rebasing is the most useful and easily visualized in the context of

a feature branching workflow from a Content perspective if you look at the workflow rebasing is changing the

base of your Branch from one commit to another making it appear as if you would created your Branch from a different

commit internally git accomplishes this by creating new comments and applying

them to the specified base it's very important to understand that even though

the branch looks the same it's composed of entirely new comments now the first thing to understand about

git rebase is that it solves the same problem as git merge

both of these commands are designed to integrate changes from one branch into another so why do we actually need

rebase so let's take a look at why we need git rebase so consider what happens

when you start working on a new feature in a dedicated Branch so then another

team member updates the Mast Branch with new comments this results in a forked history which

should be familiar to anyone who has used git as a collaboration tool the

easiest option is to merge the master Branch into the feature Branch this

creates a new merge commit in the feature branch that ties together the histories of both branches

giving you a branch structure that looks like this now merging is nice because it's a

non-destructive operation the existing branches are not changed in any way on

the other hand this also means that the feature Branch will have an externus merge commit every time you need to

incorporate Upstream changes if Master is very active this can

pollute your feature branches history quite a bit while it's possible to

mitigate the issue with Advanced git log options and it can make it hard for other

developers to understand the history of the project as an alternative to merging you can

rebase the feature Branch onto Master Branch so this moves the entire feature

Branch to begin on the tip of the master branch and effectively incorporating all of the new commits in master

but instead of using a merge commit rebasing rewrites the project history by

creating brand new comments for each commit in the original branch and the primary reason for rebasing is

to maintain a linear project history for example consider a situation where

the master branch has progressed since you started working on a feature Branch you want to get the latest updates to

the master branch in your feature Branch but you want to keep your branches

history clean so it appears as if you have been working on the latest Master Branch this gives the latter benefit of

a clean merge of your feature Branch back into the master Branch so why do we

need or why do we want to maintain a clean history the benefits of having a

clean history become tangible when performing git operations to investigate

the introduction of a regression hence rebasing hair onto the master Branch

facilitates a fast forward merge and rebasing is a common way to integrate

Upstream changes into your local repository pulling in Upstream changes with Git merge results in a super flush

merge commit every time you want to see how the project has progressed

on the other hand rebasing is like saying I want to base my changes on what everybody has already done one golden

Rune however is that you must never rebase public history you should never

rebase comments once they have been pushed to the public repository the rebase would replace the old comets with

the new ones and it would look like that part of your project history abruptly

vanished for example think about what would happen if your eBay is Master onto

your feature Branch the rebase moves all of the commits and master onto the tip of the feature the problem is that this

only happened in your Repository all of the other developers are still working with the original Master since

rebasing results in brand new comments git will think that your master branches

history has diverged from everybody else's and the only way to synchronize the two

Master branches is to merge them back together resulting in an extra merge commit and two sets of commits that

contain the same changes and the original ones are the ones from your rebase branch

needless to say this is a very confusing situation so before you run git rebase

always ask yourself is anyone else looking at this branch

if the answer is yes take your hands off the keyboard and start thinking about a non-destructive way to make your changes

for example the get reward command otherwise you're safe to rewrite history as much as you like

so now let's run a few rebase commands and see how they work so firstly let's go to this VM I have

whipped up and I have already branched out my base and we have made some changes to our

Branch using the git Branch command let's call it edureka Branch now let's

check out from here and we can see all of our previous commits also are

actually available in this new branch now you can see that this one is my

master branch and this one is my new branch and all the commits in both the branches

and add a new text file here and we're going to call it as branch

now let's move on guys and we're gonna rebase now so to rebase our Branch to master we use

git rebase master

with that you can see how under the new Branch or the new Master Branch you can

see all of the changes you have made in the rebased branch which is adiraka

branch now that was the standard rebase let's

see how we can do the same using the interactive rebase running git rebase with the hyphen I

flag begins an interactive rebasing session instead of blindly moving all of the

commits to the new base interactive rebasing gives you the opportunity to order individual commits in the process

and this lets you clean up the history by removing splitting and altering an

existing series of commits it's like git commit amend on steroids

so to rebase our Branch to master we use git rebase hyphen I

and this rebases the current Branch onto the base which is our Master but uses an

interactive rebasing session so this opens an editor where you can enter

commands for each commit to be rebased these commands determine how individual

comments will be transferred to the new base and you can also reorder the commit

listing to change the order of the commit themselves and once you have specified commands for

each commit in the rebase git will begin playing back comments applying the

rebase commands interactive rebasing gives you complete

control over what your project history looks like this affords a lot of freedom

to developers as it lets them commit a messy history while they're focused on

writing code then go back and clean it up after the fact

most developers like to use an interactive rebase to polish a feature

Branch before merging it into the main code base this gives them the opportunity to

squash insignificant comments delete obsolete ones and make sure everything

else is in order before committing to the official project history to everybody else it will look like the

entire feature was developed in a single series of well-planned commits

the real power of interactive rebasing can be seen in the history of the resulting Master Branch to everybody

else it looks like you're a brilliant developer who implemented the new feature with the perfect amount of

commits the first time around this is how interactive rebasing can keep a Project's history clean and meaningful

and rebasing can be used to change older and multiple comments committed files

and multiple messages while these are the most common applications git rebase

has additional command options that can be useful in more complex applications just git rebase d means during playback

the commit will be discarded from the final combined commit block then we have get rebase P which leaves the commit as

is it will not modify the commits message or content and will still be an

individual comment in the branches history then we have git rebase X and during

playback executes a command line shell script on each marked commit a useful

example would be to run your code basis test Suite on specific comments which

may help identify regressions during a rebase now let's talk about the configuration

options as well so there are a few rebase properties that can be set using git config and these options will alter

the get rebase output look and feel so dbase dot stat a Boolean that is set to

false by default the option toggles display of visual diff start content that shows what changed since the last

dbase and then we have rebase dot Auto squash which is a Boolean value that toggles

the Auto's cost behavior and then we have a rebase DOT missing commits check

which can be set multiple values which change rebase Behavior around missing

comments and we have one which prints uh warning output in interactive mode which wants

of remove comments then we have error which stops the rebase and prints a

removed commit warning messages and then we have ignore which is said by default

this actually ignores any missing commit warnings then we have a rebased or instruction format which is a git log

format string that will be used for formatting interactive rebase display

now let's move on guys and talk about the pitfalls of rebase so one caveat to consider when working

with Git rebase is merge conflicts may become more frequent during a rebase

workflow so this occurs if you have a long lived branch that has trade from

the master eventually you will want to rebase against master and at that time it may

contain many new comments that your branch changes May conflicts with

this is easily remedied by rebasing your branch frequently against master and

making more frequent comments the continue and about command line arguments can be passed to get rebase to

advance or reset the rebase when dealing with conflicts a more serious rebase caveat is lost

comments from interactive history rewriting running rebase in interactive

mode and executing sub commands like squash or drop will remove commits from your Branch's immediate log

at first glance this can appear as though the commits are permanently gone using git ref log these commits can be

restored and the entire rebase can be undone git rebase itself is not

seriously dangerous the real danger case arise when executing it and that's all

you really need to know and start rebasing your branches if you would prefer a clean linear

history free of unnecessary merge you should reach for git rebase when

integrating changes from another Branch it's a perfectly valid option [Music]

there is a lot of debate in the git Community about Gateway base versus get

much the question is pretty simple in a software team using git and feature

branching what is the best way to incorporate finished work back to your main line of development or you could

say your master Branch you can achieve this in two ways namely get rebase and

get merge what does get much so get much is a command that allows you to merge

branches from kit merging is a very common practice for developers where the branches are created for testing bug

fixes or any other reasons merging commits changes to another Branch it

takes the contents of a source branch and integrates it with the target branch let us understand the advantages and

disadvantages of merge and rebase first we'll discuss the advantages of merge

traceability traceability helps keeping information about the historical

existence of a feature branch and groups together all commits part of that particular feature there are a few

disadvantages so in March history can become intensely polluted by lots and lots of merge

comets also debugging using get bicycle become much harder due to the merge

commits now that we've learned the disadvantages and advantages of get much let's discuss the pros and cons of ketry

Base so the advantages are Court history always remains flat and readable in

gitry ways clean clear comment messages are as much part of the documentation of

your code base as code comments comments on your issue tracker Etc also manipulating a single commit is

extremely easy that is you can revert them back very easily there are a few disadvantages

squashing the feature down to a handful of commits can eventually hide context unless you keep around the entire

historical Branch with the development history also rebasing does not really play well with pull requests this is

because you cannot see what minor changes someone made if they have replaced rebasing can sometimes be

extremely dangerous rewriting history of shared branches is prone to teamwork

breakage this can be mitigated by doing the rebase on a copy of the feature

Branch but rebase carries the implication that competence and carefulness must be employed the largest

Advantage is that rebase sometimes requires a little more extra work using rebase to keep your feature Branch

updated requires that you resolve similar conflicts again and again yes

you can reuse recorded Solutions sometimes but merge wins here to solve

the conflicts one time and you're set moving on we'll discuss when to really use both of these commands now if you

want to see the history completely the same as it has happened you should always use much much preserves history

whereas gbase rewrites it gitmage is a default behavior when you use gitbull

use it as a default if you're not bothering about Comet history and want to avoid further problems learn to use

merge when you're always working with the big team now moving on let us understand when we can use rebase

rebasing is better to streamline a complex history here you are able to change the comment history by using

interactive replays that is hyphen I you can remove undesired commits squash two

or more commits into one or edit the commit message you can also use get rebase for creating temporary local

branches they are not necessary for public comment history and won't create any further problems and as I've already

mentioned before you can use gitree base hyphen I that is interactive for rewriting your local comment history

into a pretty one before pushing it on to a remote server your remote server can be gitlab GitHub anything of that

sort now if the feature Branch you're getting changes from is shared with other developers freebasing is usually

not recommended because rebasing process will create inconsistent repositories for individuals rebasing makes a lot

more sense that is if you're working alone or on a small team make it a habit to use rebase now moving on let us

discuss the similarities between merge and replays there is only one similarity that is both of these commands can be

used to combine two different branches I'm sure by now you must be pretty aware of that since there is only one

similarity between both of them let us understand the differences between merge and rebase and how each of them use

different techniques to combine two branches so as I've already mentioned before git

merge is a git command that allows taking the independent lines of development that get Branch created and

then eventually integrate them into a single branch on the other hand rebase is a git command that moves or combines

a sequence of commets to a new base Comet that provides easy visualization of feature branching workflow so rebase

has only one line of development or one line of work but merge on the other hand has several different lines of

development so rebase provides easy visualization of feature branching

workflow thus this is the main difference between get merge and get gbase as you can see the syntax of merge

you have firstly two lines that is git checkout feature and the second line that says get merge Master you can use

the checkout keyword to switch branches and I am switching from my feature Branch currently I'm working on my

feature branch and I want to switch to my master Branch so I'm using the line git checkout feature the second line

that says get merge Master allows me to merge the contents of my feature Branch into my master Branch or you can also

execute this entire command in one line that is get much feature Master on the

right hand side you can see the rebase syntax here also similar to the one as much there is one line that says git

checkout feature and the second line that says get rebase hyphen Eye Master and as I've already mentioned before use

git rebase hyphen I that is Interactive rewriting your local commit history into

a good one before pushing it onto your remote server so that is why we use hyphen I and the rebase keyword is used

to execute rebacing I hope both of these differences are clear now let's look at the intricate details and the

differences between git merge and get replays the first difference is based on the project history rebase provides a much

cleaner project history than get much I'm pretty sure you're clear of that by now the second difference is navigation

it is easier to navigate through the project and rebase than it is in get merge the third difference is the number

of steps merge is a one-step operation with one place to resolve conflicts and

the comments that we reachable from the branch always remain reachable rebase on the other hand is a multi-step operation

here there are more steps but each of these steps are comparatively small the

last difference is the context in git match the developer can see when the Upstream changes were incorporated into

the feature on the other hand in rebase the developer cannot see when the Upstream changes were incorporated into

the feature hence this is also a difference between get merge and get tree base moving on to the last part of

this session which one is better for which situation both and rebase can be

used to combine two branches much command just unifies your work with the commit without changing history while

rebase apply feature brand changes on top of the master branch and change the entire history if you prefer to have a

clean history then you can always choose gbase if you need to preserve the history changes then merge would

definitely be the best choice if you and your team are not familiar with or do not understand the intricacies of

Freebase then you probably shouldn't use it in this context always merge as the

safest option but if you and your team are extremely familiar with both of these options then the main decision

revolves around this do you value more a clean linear history or the traceability

of your branches in the first case go for a rebase policy in the latter go for

the merge one [Music]

get commands so we'll start with Git init command now git init is basically to turn a directory into an integer

repository now the first thing that you need you need a local repository to work with for those of you who don't know

what exactly local repository is it is basically a private copy of the whole repository of the whole shared

repository that you have and users can perform many operations with this repository such as art file remove file

rename file move file you can commit changes and you can do a lot of things with your local repository so the First

Command that we are going to discuss is basically the git init command which will turn a directory into an empty git repository so for that I'll open my

Virtual Machine to show you practically if you are facing any issues in installing git you can go ahead and refer my git installation video there

also I have installed git incentives version 6.5 so for now let me open my Virtual Machine and I'll show you there

so guys this is my virtual machine where I have installed git if you want to see the version of git that you have

installed just type in here get hyphen hyphen version and it is 2.7.2 in my case case now let's go ahead and create

the directory first so I'm going to name my directory as edureka hyphen Repository

now I'm going to move into this directory so I'll just type in here serial array card repo so now I have a

directory I want to convert it into a local git repository which is very easy all I have to do is type in git init and

here we go so initialized empty git repository in home address so let's go back to our slides again and

we'll see which command we are going to discuss next let's talk about git add now so git ad

basically adds all the files to the staging area for git now git doesn't track each and every modified file so

whenever you do the commit operation git looks for the files present in the staging area only those files present in

the staging area are considered for commit and not all the modified files so the first thing you need to do is put

all the modified files all the changes that you have done into the staging area and then from staging area you can go

ahead and make a comment right so the first thing is to add your files to the staging area which is pretty easy all I

have to do is type in git add in the file name so let me just quickly open my Virtual Machine and show you there this

is my virtual machine again guys and first let me just create a file here let the name of my file be uh edureka.py

yeah let me just go ahead and type something in this uh dot py file

so I'll just type in here a equal to 10 a small code be equal to 20 and I'm

going to print a plus b pretty basic save it and go ahead and close it right

now what I need to do is add the changes that I've done into the staging area right so all I need to do is type in

here get add and the name of the file that is edureka.py so I've successfully done that if I want

to add all the files to the staging area then I can go ahead and type git add and a period right so this will add all the

files that are there in my local repository to the staging area so this was all about git add let me just

quickly open my slides and we'll see which is the next command we are going to discuss now we're going to see the

git commit command so git commit basically records the changes made to the files in a local repository so you

have all the modified files in the staging area so when you make a commit it'll basically record the changes that

you have made to the files in that local repository and for easy reference each commit has a unique ID and it is a good

practice to always give a commit message right so that helps you in understanding what change was done in this particular

comment right so I'll just quickly open my Virtual Machine and I'll show you how you can commit changes to the local repository this is my virtual machine

again guys so I'm just going to type in here get commit hyphen M and then commit

message so I'll just type in here first comet right so I've done that successfully two

files changed five insertions create mode one double zero six double four at arraycar dot py right so this is the log

for it let me just clear the terminal and go back to my slides again let's talk about a git status now

so git status Returns the current state of the repository git status will return the current working Branch if a file is

in staging area but not committed it shows with Git status or if there are no changes it will return nothing to commit

working directories clean right so let me just open uh my virtual machine again and I'll show you practically how this

works all right so if I go ahead and type here get status so it says on Branch Master nothing to

commit working directory clean so it gives me the name of my branch and what is there anything that we need to commit

or something that is already there in the staging area but I have not committed that and the working directory

is clean right now so whatever changes that I've made in my file are there in the staging area and from the staging

area I've made a comment as well right so let me just do one thing make some changes in the file add it to the

staging area but I won't commit it right so let's see what will be the status then so for that I'll create one more

file let that be welcome.pui I'll edit this using G edit

you can use whatever editor you want I'll just type in here print welcome to edureka close the parenthesis

save it and close it now I'm gonna add this file to the staging area so I'm

just going to type in here git add I can go ahead and type a period as well that will add all the files or I can give the

name of the file that is welcome.py now if I type git status let us see what

will come so it says new file welcome.py changes to be committed right so these changes are present in the staging area

but I have not committed these changes and it gives me the name of my Branch as well so I hope you have understood what

exactly git status is so I'll just quickly open my slides again and see which command we are going to discuss next let's talk about git config now now

with Git there are many configurations and settings possible so git config is basically how to assign these settings

now two important settings are username and user email address these values set what email address and name commits will

be from on a local computer and it's very easy to do that let me just quickly open my VM and I'll show you that

now there are two things either I can run the git config command globally or I

can run the git config command on the current repository settings so let me show you how you can do it globally so

I'll just type in here git config hyphen hyphen Global user dot name and the name

of the user so let it be subham and now I can go ahead and add the email

address as well so I just need to type in here user dot email and I'll type the

email address and the parenthesis in the double quotes right so this is done

and this was done globally now if I want to execute this on the current repository what I'll type in git config

I won't use hyphen iPhone Global I will directly go ahead and type user dot name

so let it be my name I say u r a b h and I'm gonna add my email address as

well which is sort of at the rate eduraycar.com done guys it's that easy now let me

quickly open my slides again and we'll see which command we are going to execute next so we've seen how a git

config Works let's discuss about git branching now now before I proceed let me tell you that there will be a

separate tutorial video on git branching where we are going to discuss in detail about it here I'll just give you a quick

introduction to branching so the first question is why do we need Branch right so branches are used to

create another line of development and by default git has a master Branch so usually a branch is created to work on a

new feature so if you're working on a web application you might want to work on the card service right so for that

you'll create a new feature Branch or any name that you want to give to your branch and once the feature is completed it is merged back with the master branch

and we delete the feature branch and every branch is referenced by head which points to the latest comment in the

branch whenever you make a commit head is updated with the latest commit let me just quickly open my Virtual Machine and

over there we're going to see how to create a branch how to delete a branch how to rename how to merge and how to

check out into a different branch this is my virtual machine again guys and uh let me just tell you how you can

see all the branches so just type in here get branch and uh

and it will list all the branches that are already there so there's only one master Branch present right now so let

me just quickly go ahead and create a branch see there are two ways to create a branch one way is to create a branch

and then you can check into that Branch another is you just want to create it but you still want to be in your master Branch or your current working Branch so

for that uh if you don't want to check in that Branch just type in here get branch and give a name to your brand so

let that be feature right and if I go ahead and type git Branch now I can see that there are two branches master and

feature right now I can go ahead and delete this Branch as well but let me just quickly

uh move into this Branch first so I'll just type in here git checkout

and the name of my branch that is featured and this will Point towards my feature

Branch now I'm in my feature Branch right so let me just make some changes in the files that are there in my

edureka repo so you can see that I have couple of files data rica.py and welcome.py so let me just make some

change in let it be at arraycar.py right so I'll just quickly type in here G edit

again you can use whatever editor that you want edureka.py and I'll just add one more

variable here let that be 30 and I'll print one more value

that is C plus a plus b pretty basic save it and then go back

so in my feature Branch I have made changes to my editoraker.py file right

so what will be my next step is to add these changes add the modified file in the staging area so I'll just type in

here git add on a period because I want all the changes that I've made to my files should be there in the staging

area so git add and period will add all the files now I want to commit these

changes so I'll just type in here git commit hyphen M and a message feature

commit right so results now I've done all the changes and I've added those changes to the staging area and then

finally I've made a comment as well so it says four files change 10 insertions create mode welcome.py welcome.py let me

just clear my terminal now so if I want to merge my feature Branch with the master Branch now first I need to check

in with my master Branch for that I'll type in here get check out master

now it says switching to Branch master from feature now I'm in my master Branch so now I'm gonna type in here git merge

feature and here we go so you can see that I've merged my

feature Branch with that of the master Branch it's that easy guys let me clear my terminal now so what I'll do I'll

quickly create one more branch and and I'll show you again uh how these things are happening because there might be

some confusion still so I'll just show you again how to create a branch how to check into that Branch how to make some

changes in your local repository add that to the staging area make a commit and then merge it with the master branch

so now uh what I'm gonna do is I'm going to delete my feature Branch because I've

told you earlier as well once I've merged all my changes to my master Branch there's no point in keeping my feature Branch as of now I delete that

branch and there are a couple of options to do that the safest one is what I'm gonna tell you right so I'll just

quickly uh type in here uh git Branch hyphen D and the name of your

branch which in my case is feature

so it has deleted the branch feature now there's one more way to delete a branch instead of hyphen D you can just type in

here get Branch hyphen uppercase D and then the name of the branch but this is

not the preferred option and it's not safe as well because if you have some unmerged changes then also it will

delete the branch whereas if you type in here a lowercase D it will show you a error that will say that you have some

unmerged changes right so always prefer a hyphen small d uh and then feature

hyphen small D another name of the branch that you want to delete right so now I'm going to show you one more way

to create a branch which is called the git checkout hyphen B and you can give

the name to your branch right so probably I want to name my Branch again as a new let the name of my Branch be

new so git check out hyphen B new so this will not only create a branch called new but it will also check into

that Branch right so we have also checked switch to a new branch that is new and over here also if I want to make

changes to my file I can go ahead and do that all I have to do is use any editor let me uh edit say welcome dot py file

welcome to edureka I forgot to close the quotation so

that's the change I'm gonna do in this file right and then I'm gonna get add period now get commit

I made a mistake there so I'll just type in here get commit hyphen m and uh let it be new to my message all

right so it is done now now what I'm going to do is I'm gonna show you how to connect to a remote

repository so I have a remote repository the GitHub account I've created a repository there so if I want to connect

to that repository what I need to do is I need to add that origin so I'll just type in here gets remote add origin

and the SSH link that I will just show you where you can find it so this is my

GitHub account guys and this is my repository that is git commands if I want to copy the SSH link I can just go

ahead copy this and paste it here so we have successfully added the origin

now let me just clear my terminal so we have connected a local repository with the GitHub account if you face any

issues with that you can go ahead and refer my git installation video so we have seen how branching Works in git let

us move forward and we'll see how to work with remote repository so over here I've already told you how to add an origin right it's a git remote add

origin and what we did we connected a local repository with a remote repository so The edirecta Hyphen repo

is my local repository and a remote repository I just showed you in my GitHub account so we have already done

this git remote ad origin so now we're going to see how to clone a remote repository so to create a local working

copy of an existing remote repository we'll use git clone to copy and download the repository to a local computer

cloning is the equivalent of git init when working with a remote repository

git will create a directory locally with all the files in repository history

so I'll just open my Virtual Machine and I'll quickly show you how you can do that all right so what I'm going to do

is I'm first gonna create one more directory so let that be mkdir

and uh the name of the directory Let It Be git hyphen repo right and let's move

into this git hyphen Repository so first thing I'm gonna do is git clone

git clone and the SSH link so SSH link I've already shown you where

you can find it in your GitHub account just type in here get clone and the remote repository URL the SSH link that

I was talking about and let me show you how you can do that and here we go so it says cloning into git commands

so let me see what are the files that are there now so it says git commands let me go to this git commands

so here we have ederica and readme and here also you can notice that we have ederica and readme so I've successfully

done that let me clear my terminal and now uh I can go ahead and

pull whatever changes I made to the file that is there in my GitHub account so let me just show you how to do that

suppose if I have a file edureka and let me just make some changes here

I'll just type in here my name is saurabh

so let it be like this so let's come and change this

so yeah now it'll be reflected in my Repository now what I want is I want all the

changes that have been made in my remote repository in my local machine so for that I'll just type in here git pull

origin master and here we go so it says it has successfully done that

that means we have pulled all the files that are there in a report repository so if I check so I have here etereka and

readme.md files so if I go ahead and type in G edit edureka

let's see or or right so I can see that the changes have been reflected my name is saurabh which was something new that

I added to my file right so this is all about the pull operation now let us see how you can push changes to the remote

repository right so if I go ahead and make some changes or if I create a file

let's say I create a file touch get Dot py let that be get one dot py right so

if I make some changes to this particular file I'll write here print

I love git save it close it right so now before we

push to the remote repository we need to add these changes to the staging area we

need to make a commit and then only we can push it right so first thing add the file name so I'll just type in here get

add I can go ahead and type here if period that will add all the changes made to the files in the staging area

all the files in the staging area basically modified files or I can type the name of my file which in my case is

git1.py so I'll just type in here get add get1.py and now I'm going to commit

the changes so I'll just type in here git commit hyphen m

give a message so I'll just type in here remote and it has done successfully so now I

can go ahead and push the changes to my remote repository so I'll just type in

here get push origin master and here we go

so it say that it's done successfully let us go back to our GitHub account and we'll see whether those changes are reflected there and or not so I'll just

go ahead and run this basically reload it so you can see that get1.py file is

added here right so let me just quickly go to the slides again and we'll see what all topics we have already covered

so we have seen how to perform git clone we have seen how to pull changes from the remote repository to the local computer we have seen how to push the

local government to the remote repository now let's talk about few Advanced git commands focus on git stash

git log gets rebase and get revert so I'll quickly open my Virtual Machine and I will explain you there only so guys

let's discuss about get stash first so to save changes made when they are not in a state to commit them to a

repository you use gitstash this will store the work and give a clean working directory if I have to give you an

example imagine when you're working on a new feature that's not complete but an urgent bug needs attention so that's

when you can use get stash let me just quickly show you how you can use it so the first thing I'm going to do is

create a new file so let the name of my file be stash Dot py

and I'm gonna make some changes to the stash.py I'll just type in here print

stash is cool save it close it and yeah now what I

need to do is add this to the staging area so I'll just type in here get add

and now let's see the git status so my git status says that we have new file which are there in the staging area

that is stash.py right changes to be committed so these are the changes that

I need to commit So currently it is pretty dirty so what I can do is I can put all the uncommitted changes to a

stash let me show you how can do that get stash hyphen new and here we go now

go ahead and type git status nothing to do working directory clean so it has converted my dirty directory to a

pretty clean one right so with the help of git stash right now the modification stashed away by this command can be

listed with Git stash list so I'll just type in here get stash list here we go

and if you want to inspect it so you can just go ahead and type git stash and show

so you can inspect it here right Let Me Clear My terminal now so now I'm done

I've fixed my bug and now I'm ready to you know work on what exactly I was doing before fixing that bug right so

what I can do is I just want to type in here get stash apply and here we go

and now if I type git status you can see that I again got the dirty directory back

so guys we have seen how gitstash works now what I'm going to do I'm going to show you how a git log and git reward

works right so uh for that I'll quickly create a new repository so let me do

that I'll just type in here mkdir git hyphen log so let that be the

name of my directory and I'll go into that directory yeah so now I'm going to

initialize it so I'm just going to type in here get init it has been done now I

will uh create few files here uh let that be edureka1 Dot py and I'll just

type in here print Master new technologies

at edu breaker all right save it

and close it now I'm going to add it here git add and then finally committed

hyphen m just give a message and it is done now so what I did I created a directory

initialized it then what I did I made a file inside that in that file I added

that to my staging area that file has been added to my staging area and then finally I made a commit right so till

now there is no problem now if I type here get log let me show you what comes

so you can see that when I type git log there's a commit hash right there is a commit that has been made by authorship

on this particular date and type right so this is what basically git log is it shows the chronological comment history

for a repository this helps you give context in history for a repository and it is available immediately on a

recently cloned repository to see the history as well now I can provide parameters here as well for example I

can type in here get log hyphen before and give a date here right similarly I

can tell the git lock to show me the commits based on the author right so if I type in here get log hyphen hyphen

author and the name of the author then it will show the logs based on the commit made by the author right

similarly if I type in here hyphen hyphen before right and give a date so it will

give me according to the date as well so this is how git log Works guys so now what I'm going to show you is how to

revert to the previous commit right so for that let me just make some changes to my file again so the name of my file

is edirecta1.py so I'll just type in here G edit edureka1.py and I'll type in here one

more print statement that says don't just simply

learn it Master it

save it close it now I'm going to add it to my staging area and finally I'm gonna

commit it hyphen M last commit all right so this

is done now now what I have to do is suppose I have made some changes I'm gonna go back to the previous change

right so I've made some change in my code I've done the commit as well but I want to go back to the previous commit

right so for that what we use we use git revert so let me show you how you can do that so in order to go back to the

previous comment what I have to do is just type in here get log hyphen hyphen one line this will give me one line so

this was my last comment so I want to go back right so for that I'm just going to type in here get revert and the hash

comment hash and here we go so it says that revert last commit so it

has already done that now if I go ahead and check my EDU recap.1.py file so it

only says Master new technologies at edureka so this is how you can even perform the revert operation just by

using in the uh commit hash so with that you can go ahead and revert back to the

previous commit so I hope this is clear now let me clear my terminal so now what I can do is I can go ahead and revert to

the last commit as well instead of just typing the commit hash if I just type in here get reward head

and if I quit so it says that I've rolled back to the previous version of my file so if I go ahead and click on LS

command and if I type in here cat at eureka1.py so I've gone back don't just

simply learn it Master it so my last commit was basically revert operation right so whatever file changes that have

been done after gitreward will be reflected as a commit itself right so I

hope you're getting my point so what I did first I created an empty repository then in that I created a file editoraker

one dot py made a couple of changes made two commits then I used get revert with the hash commit to go back to the

previous comment right so with that I got only Master new technologies etraca this one I didn't get at that time then

what I did I showed you get revert head which will basically take the last comment right it will reverse the last

comment and the last comment was nothing but the revert operation that happened that actually made my file I'd rake

a1.py to go back to the previous version so that was my last comment right so because of that it has revert back to

the last comment this is why you can see that edrica1.py file has both Master new technologies at the direct and don't

just simply learn it Master it right so I hope I'm clear with Git log and git reward it's very easy guys there's not a

lot of things to understand here it's very basic so let's talk about rebase now so basically rebase is again one

more way of combining the work between different branches so what rebasing does it will take a set of comments copy them

and store them outside your repository the advantage of rebasing is that it can be used to make linear sequence of

comments All right so the commit log or history of the repository stays clean if rebasing is done right so let me just

show you how it is done if I just type in here get rebase master let me just go ahead and do that git rebase master

so the current Master is up to date that's what the message pops up so basically this command will move all

our work from current Branch to the master Branch they look like as if they are developed sequentially but they are

developed parallely all right so let me just give you a better example of this so I'll just

create a branch here git branch say

and Eureka 0 1 0 is the name of my branch and uh

yeah so if I want to go to this branch check out and eureka010

yeah I went to that branch and if I make changes to

indirect1.py I'm going to remove this print statement save it close it clean my terminal and

I'm gonna add it get add period and get

commit hyphen m rebase all right so this is done now and now

what I can do is I can go ahead and type in here get rebase master

so it says that current Branch area 0 1 0 is up to date so this is how you can perform rebasing so let me quickly go

back to my slides so we have talked about git stash we talked about git log git rebase and git revert

Jenkins get integration first we will start with what exactly is Jenkins then we will talk about the

various advantages of integrating git with Jenkins finally we will see a demo which will

show you how do you integrate git with Jenkins I hope I am clear with the agenda guys it is pretty much simple and

just a reminder in case you have not yet subscribed to our YouTube Channel please do subscribe and also hit the Bell icon

so that you never miss an update from edureka and also if you're someone who's looking for a course in devops you can

check the link for that course in the description box below now let us talk about the definition of Jenkins

so Jenkins is an open source automation tool written in Java with plugins built

for continuous integration purpose Jenkins is used to build and test your software projects continuously making it

easier for developers to integrate changes to the project and also making it easier for the users to obtain a

fresh build it also allows you to continuously deliver your software by integrating with a large number of

testing and deployment Technologies with Jenkins organizations can accelerate the

software development process through automation Jenkins integrates development lifecycle processes of all

kinds including build document test package stage deploy static analysis and

much more Jenkins achieves continuous integration with the help of plugins plugins allow

the integration of various devops stages if you want to integrate a particular tool you just need to install the

plugins that are required for that tool for example the various plugins that are supported by Jenkins include get Maven 2

project Amazon ec2 HTML publisher and so on so this was the definition of git now

if you talk about the advantages of git as already discussed it is an open source tool and it has a great Community

Support it is very much easy to install and almost has 1000 plus plugins to ease

your work if a plugin does not exist you can code it and share it with the community

it is free of cost and it is built with Java and hence it is portable to all the

major platforms so this was all about the Jenkins now let us talk about why should we

integrate git with Jenkins or why they are used together so as already discussed git is a source

control manager and that's where you keep your source code for tracking all the code changes that happen over the

time and to Baseline the versions when they are ready for release Jenkins on the other hand is a

continuous integration solution it was devised to automate most of the tasks that a developer has to do while

developing a new application without a CI solution a developer has to spend much time doing these repetitive

non-productive tasks so let us take an example of a new feature in a web application so what happens here is a

developer will be identified and assigned the task so he takes the existing code base from the source code

say for example git makes its changes carries out unit testing and shows the

code quality manually and checks the new code back into the gate then he has to

build the code deploy it into the integration instance run integration testing and once the change seems

satisfactory he raises a request for production deployment now if we only had

a minion to take care of this testing building quality checks and deployment sections the poor developer could have

focused better on the stuff that is really good at that is implementing and improving the features logic

so this minion is nothing but Jenkins it acts as an orchestrator which runs

all of these activities once the change is checked into the source control and thus gives a quick feedback to the

developer whether the changes he has made are good enough for the production deployment or not so Jenkins is

extremely flexible open source and has a lot of plugins that can do almost anything that you wish to so these were

the reasons of using Gateway Jenkins now next comes the question of how do they work together

so here I'll be showing you one demo and these are the steps that are covered in

this demo so first we create a sample program so you can create any program that you want be it a Python program or

a Java program or any other program then in the second step we create a Jenkins job and once we create the Jenkins job

we add the program that is created in the first step to the GitHub so for that we create a GitHub repository a public

GitHub repository and upload the code on that repository using git and in The

Next Step you add git plugin in Jenkins and finally we configure the Jenkins job

based on the GitHub repository so let me just show you how do you do this

so what we need to do here is we have to create a directory on your desktop and

you have to create one python file here so the name of the file is say for example anything that you want on in

this case I've written my file okay so this is basically a simple python file with a very simple code print hello

world that's it so what we need to do here is you can open your git bash here

so first let us initialize an entry get repository here using the command get init

it's an empty repository has been initialized here in this demo folder so now let us just check the status

here using the command get status so as you can see here this file it is

in red color my file dot porn so red color means here that it has not been added so just to add the file you can

use the command get add and the dot you can also specify the name of the file here like my file.view

app but if there are multiple files that you need to add then you can use the dot symbol

so the file has been added and now let us just verify this okay so as you can see here previously

it was red and now it is green here so after adding the file now let us just commit these changes

for that the command is git commit hyphen M any message if you want add my

file so we have now committed this changes so

what we need to do here is we have to push this my file.py on the GitHub repository so for that let me just go to

my GitHub account okay okay guys so what we need to do here is

we have to create a public repository for that you have to click on this repositories on your GitHub main page

and you have to click on new so once you click on new you can name

the repository say for example git Jenkins integration

or whatever name that you want and the description is optional here and if you want you can initialize this repository

with a readme and click on create Repository so this has been created here

so guys now we have created a public repository on GitHub and now our task is

to upload this file called myfile.py onto the public repository on GitHub

for that we will have to first connect to it and the command for that is get remote add origin

and the link of that repository and how do you get that link so when you create this repository here

you get the link okay so just copy it and paste it here

so this has been added and now we can push our content there so for that the command is git push

hyphen U origin and master

here you can see if you just refresh this page okay so my file dot py here it

has been added onto our public repository on GitHub so now what our job

is we have to create a Jenkins job first login onto your Jenkins homepage

enter the username and the password so now what we'll do here is we will

create here a new job Jenkins get demo and choose the freestyle project

click on OK see here what you need to specify the

source code Management in the source code management okay before that you have to download

the get plugin so how do you do that go to manage Jenkins here as you can see

left hand click on manage plugins okay if you already have git plugin

installed on your Jenkins you can click on install it here

so if it is already downloaded yeah as you can see here on my system it has already been downloaded you can find it

here else if it is not available like if it is not installed you can click on available search here by typing git and

download so this is how you download the get plugin and going back to our project

let us just configure this now

okay so this was a project that we created Jenkins git demo so let us just

configure this project

so in the source code management section here you must click on get and you enter the git repository there

copy this this link and you paste it here

click on apply it has been added and here in the build

trigger section we have to click on policy so here we have to enter the

Chrome expression periodic scheduling of Jenkins job is done through pole sem so

what you need to do here is in my case I'm typing here 5 asterisks separated by

space I hope you guys are aware of the KRON syntax so five asterisks here mean that our

Jenkins job will run every single minute so every single minute our Jenkins job will check for the source code change if

there is any change it will run like it will build the job it will build the code this is the meaning of that click on

apply yeah and now since we have a python code so what we will do here we

will add a build step since guys I'm using here Windows operating system so I'm clicking on execute Windows batch

command if you are using Linux or Mac OS you can execute shell script okay so

click on here so since I have a python code there so I will type python like

the command to run the python file my file dot the file okay fair enough

click on apply save so now we will build this project

okay as you can see here our build was successful so this was the hello world thing that

we had written in our program and here it shows success okay so now what we

will try to do is we will do some changes there in the my file we will add

a few lines and then we will see what happens Ctrl C say for example hello world now we will

say how are you save it

okay so now we have to push these changes onto the public repository of GitHub so we will do that through git so

now let's check the status first okay so modified as you can see here the

file has been modified and that has been shown here so for that we will first add

and now let's check the status so now we will commit this file

some message if you want modified my file

press enter so now we have to push these changes so for that the command is get push

hyphen U origin master

so these changes have been pushed onto the GitHub account so let me just refresh this page

okay so as you can see here modified my file this was the commit message that we had written so our changes have been

reflected here if you can see here print how are you we had added this line if you remember

so now we will see what happens to Jenkins now

so here we will click on build now

so if you click on Console output here okay started by an SCM change so since

there was change in the source code that triggered the Jenkins job so here you can see Hello World

and how are you both the lines have been displayed here and it is success

[Music] foreign [Music]

systems can decrease are spent on maintaining tool chains for these systems to work it's essential

for you to catch bugs early in the development cycle ensuring all the code deployed to the production complies with

the code standards you established hence CI CD pipelines are becoming increasingly crucial in the software

development process we shall discuss how to build a simple CI CD pipeline using

gitlab CI CD so that you can accelerate your devops transformation no matter where you are in your devops journey

before we begin let's take a look at our agenda for this session by introducing you to all the concepts of CI and CD

along with its basic workflow and stages following that we shall talk about a CI

CD pipeline its architecture and significance then we shall dive straight into understanding gitlab CI CD and how

it works to implement and maintain a CI CD system to best serve your organization's needs we'll learn to

install and set up a gitlab runner to our systems and finally I shall demonstrate a Hands-On that I'm going to

build and run a very simple pipeline using gitlab CI CD Now The Continuous

methodologies of software development are based on automating the execution of scripts to minimize the chance of

introducing errors while developing applications they require less human intervention or even no intervention at

all from the development of new code until its deployment it involves continuous building and deploying code

changes at every small iteration reducing the chance of deploying new code based on budget or failed previous

versions there are three main approaches to this methodology each of them to be applied according to Wordpress your

strategy continuous integration consider multiple developers working on an app this application has its code stored in

each of their respective local repositories But ultimately there has to be one code isn't it therefore all the

involved developers push their code changes to a common remote repository to the code this happens every day multiple

times a day till the code is complete this practice is known as continuous integration for every change submitted

to an application it's built and tested automatically and continuously ensuring the introduce changes pass all tests

guidelines and Code Compliance standards you established for your app eventually

decreasing the chance of introducing errors in your application continuous delivery this is a Step Beyond

continuous integration your application is not only Built and tested at every code change pushed to the code base but

as an additional step it is also deployed continuously though the deployments are triggered manually

this method ensures that the code is checked automatically but requires human intervention to manually and

strategically trigger the deployment of the changes continuous deployment this

is also a Step Beyond continuous integration similar to continuous delivery the difference is that instead

of deploying your application manually you set it to be deployed automatically this does not require human intervention

at all to have your application deployed if we take a deeper look into the basic workflow we can see each stage of devops

lifecycle so you basically verify package and release your application using the CI CD

pipeline this has quite a few advantages in the deployment and production environment including but not limited to increased

speed of operation communication between teams regular feedback due to frequent

deployment Cycles easier detection of Errors resulting in early recovery now

that we have some idea about CI CD let's dive a little deeper into the element which makes it all possible pipelines so

what are pipelines pipelines are basically the fundamental building blocks of cicd and are hence the

backbone of modern devops environment a CI CD pipeline Bridges the gap between development and operations team by

automating the building testing and deployment of applications it basically helps to automate a series of steps or

stages in your software development process such as initiating code builds running tests deploying and staging

production environments and everything in between it provides a more standardized approach to feedback loops

reducing errors and paces of iterations of development now there are different stages in pipeline implementation which

lays a strong foundation for the continuous everything we also desperately seek to achieve at number

one is commit in this stage new codes are integrated to the base code you have a feedback loop to help check

the code quality and a series of unit tests and Sanity tests for the same this

is not to be mistaken with the testing stage that comes later in the pipeline implementation

next up is build once the code is tested and integrated in the next stage the

code is built this is where the source code is converted into an executable form that can be run on a machine

the testing stage in the testing stage the alpha and beta deployments take place the former checks the integration

between various builds at hand while the latter to check if the app is working before moving it to production the

deployment stage once the build has gone through all the test scenarios successfully it is ready to be deployed

to the production stage you then finally move to the fourth and final stage and deploy your application and your app

goes live for the end user environment that is your consumers now there are a couple of tools popular for being able

to automate this entire process two big names you'd often hear are Jenkins and gitlab CI CD to know more

about Jenkins you can check out the extensive Jenkins playlist on our Channel but today we are going to tackle

the big name that is gitlab cicd gitlab cicd is a powerful tool into gitlab that

allows you to apply all the continuous methods that is continuous integration continuous delivery and continuous

deployment to your software with no third party application or integration needed to use gitlab CI CD all you need

is an application code base hosted in a git repository and for your build test

and deployment scripts to be specified in a file called gitlab ci.yml located

in the root path of your repository in this file you can Define scripts you want to run Define include and cache

dependencies choose commands that you want to run in sequence and those you want to run in parallel Define where you want to deploy

your application and specify whether you want to run your scripts automatically or trigger any of them manually once

you're familiar with gitlab cicd you can add more steps in the configuration file to add scripts to the file you'll need

to organize them in a sequence that suits your application and are in accordance with the tests you wish to

perform to visualize this process imagine that all the scripts you add to the configuration file are same as the

commands that you run on the terminal on your computer so here is an example of the simplest pipeline configuration that

can be using a minimal Docker image based on Alpine Linux you have to build

build a build something and build B build something else this pipeline will

run everything in build stage concurrently and once all of those finish it will run everything in the

test stage the same way and then deploy A and B now this pipeline needs to

become configured using a yaml file called as the gitlab cicd.yml within each project the gitlab

ci.yml file defines the structure and Order of the pipelines and determines what to execute and what decisions to

make when specific conditions are encountered say for example when a process succeeds or fails here's the

yaml file for the same pipeline diagram we saw before you can see all the different stages in the pipeline that is

built test and deploy and each stage has a script to be printed to depict what's

going on in that particular stage fairly simple isn't it now note that it's a

very rudimentary Pipeline and definitely not the most efficient and if you have lots of steps it can grow quite complex

but it's easy to maintain there are other pipelines for that such as direct acyclic graphs and parents child

pipelines but since we are just starting out a basic pipeline is what we are going to understand and Implement later

on in the the session once you've added the configuration file which is the yaml file to your repository gitlab will

detect it and run your scripts with the tool called gitlab Runner which was similar to your terminal the scripts are

grouped into jobs and together they compose a pipeline it is designed to run on gnu Linux Mac OS and Windows

operating systems gitlab cicd not only executes the jobs that you've set but

also shows you what exactly happens during the execution just as you would see in your terminal you create the

strategy for your application and gitlab runs the pipeline for you according to what you've defined your pipeline status

is also displayed by gitlab at the end if anything goes wrong you can easily roll back all the changes consider the

following example for how gitlab CI CD fits in a common deployment workflow assume that you have discussed a code

implementation in an issue and worked locally on your proposed changes once

you push your commits to a future branch in a remote repository in gitlab the CI CD pipeline set for your project is

triggered by doing so gitlab CI CD runs automated scripts that can be sequential

or parallel to build and test your application and preview the changes per merge request once you're happy with

your implementation you can get your code reviewed and approved merge the feature Branch into the default Branch

gitlab cicd deploys your changes automatically to a production environment and finally you and your

team can easily roll back if something goes wrong gitlab cicd is capable of doing a lot more now that you know about

gitlab cicd let's install and set up what we need to be able to run our pipeline so first things first let's

install gitlab I'm running Ubuntu 19 on my virtual machine so let's open the

terminal first we need to install a few packages to install the git lab and those are the open SSH server and

postfix mail so what which can be installed using the following command

you have to configure the internet and set the system mail as follows here you have to set the system mail

the required packages have now been installed in the next step we have to install gitlab and there are various

methods to install it we can download a dbn package directly onto our systems so

in order to download that you need to go to the official site

and get the link directly based on your distribution

so now let's just copy and paste it onto our terminal

and hit enter after executing this command you'll have to navigate to the directory where it's stored and then

install it

as you can see the dbn package is available and you can install it using the following command

now this might take about five minutes based on your system configuration and for installing it you need 4GB of ram at

least so gitlab is partially installed as you can see here it's unable to detect a

valid host name for your instance for this you have to convert a URL we're

going to edit this file now and add your host name

so after adding your host name just save and quit the file now you can start your gitlab instance by running the following

command so as you can see over here the gitlab instance has been configured in this machine and you can now access it

from the browser go to gitlab.com and create an account I've already created mine to do that all

you have to do is fill in some information in the given field it contains nothing but your name email and

a password check the first box which is the terms and conditions acceptance checkbox and also confirm that you are

not a robot by any means click on the green register button and you're done this will send the confirmation mail to

your email ID open it and confirm your gitlab sign up you can then sign in with your email and password to gitlab so

here is what mine looks like so once you're logged into your account you should see something like this click

on the new project button over here so we are going to create a very simple project the one from earlier just to

demonstrate the basic functionality of gitlab so go ahead and click on new project and populate this field with

something I'll type at Eureka Pipeline and right here below you have the possibility of

selecting the visibility you can make it public if you want but I will leave it private because I'm just playing around

so go ahead and create the project once you're ready you will have an empty repository like this there's nothing

inside it so the first thing that we are going to do is create a new file the file will be called Master slash dot

gitlab hyphen CI dot yml this is how the file should be named and this is the

convention it will automatically recognize so I'll paste my pipeline code inside here and what we are trying to do

here is to create three basic stages there is a build stage a test stage and a deploy stage in order to get something

to run you need to Define some steps in each of these stages this is just to demonstrate that you can enter multiple

commands one after another in a stage so now we have this file let's commit

and see if the syntax that we have over here is valid now we're getting this message over here that says gitlab

configuration is valid congrats now if you go to the project overview you will see here that something has started to

build this is the icon that shows the pipeline has started and I did not need to configure anything in particular just

by having this file here and no other files gitlab started building this pipeline as instructed and if I click on

this icon I'll actually be able to see the pipeline how it looks within three stages

now this will take a minute to start a Docker image and to actually execute it the first stage succeeded this goes down

to the second stage now you will see that the second stage is running as well

now let's see if the pipeline is successful you can easily see the pipelines that are currently running if

you go on over here on the left panel and click on CI CD Pipelines you can see that the current pipeline is

running the stage it's in and how much it still needs to be in order to complete now I just refreshed the page

and we see that it has run all the stages successfully foreign [Music]

ERS comes a level of agility never before experienced in the world of

business in seconds administrators can roll out deployments for a multitude of

services and when those businesses need to scale down those containers they can turn to Docker swarm or communities but

what exactly happens when even a standard container workflow cannot keep

up with the ever-growing demand of business you would be hard-pressed to find a more efficient means of deploying

and scaling a container cluster and although that may very well be true it

does not mean the development cycle cannot be improved git Ops is a method of workflow that was conceptualized by

everyone who successfully did infrastructure as code and today we shall talk all about it let's discuss

the agenda for today starting out we'll get familiar with the term git Ops what

it is and why it is required its main principles and features then we shall

move on to the working of git Ops and discuss each stage of the process moving

on we shall discuss the advantages of githubs so without Much Ado let's get

started so why git Ops previously companies usually had separate

operations and development teams and unfortunately they did not talk to each

other too much luckily we have devops which tries to address this by ensuring

increased collaboration so that operations and development teams work closely together in an Ideal World it

meant that you have the same people who are both writing the applications and operating the applications it means

automating as much as possible to reduce the need for manual human input when

operating applications so devops was a cultural change now with Git Ops it

takes it to the next level where teams have an operating model where they not only Define infrastructure as code but

also make deployments and changes to infrastructure as code by submitting pull requests that people can eventually

review and approve collectively with githubs you may get ubic wishes to have

continuous integration on a continuous delivery server when deploying your application now once you submit a change

the server will pick it up immediately pull the changes execute any automatic

tests execute any quality checks and eventually build the artifact this

artifact is then published to an artifact repository and eventually deployed to infrastructure by applying

git Ops there is a source of tools for both your infrastructure and application code allowing development teams to

increase velocity and eventually improve system reliability with all of this said

what exactly is git Ops on the surface it is quite simple git Ops is centered

around using a Version Control System such as git to house all information

documentation and code for a cluster deployment and then use automated

directors to deploy changes to the cluster however once you dig past the surface you discover that it's far more

complex than that git Ops is the operational practice which users get as a single source of Truth it is to be

noted that Source control repository on git becomes the source of truth and not

the actual servers or the Clusters this is nothing but having your infrastructure as a code which means all

your infrastructure setup is inside a code base it also includes the automation of deployments rollbacks and

much more git repository can be leveraged for version controlling system peer reviewing system automating and

deploying process for the production environment using git itself developers

are now doing continuous delivery and automated pipelines Additionally the web

hooks from from the git can be leveraged to push these configurations into the development and test environments so

once you merge that particular pull request onto the main branch the deployment to production eventually

happens on the other hand git Ops also allows automating everything using

pipelines and deploying that to production once you merge the code into your production Branch it is called get

Ops because all the configurations are managed in the get repository many developers deploy the infrastructure

code as well as a part of the automation process by using only one repository for

an application or a service and have a separate repository for each of them now

let's say you have three environments namely development test and production there should also be three branches that

are named the same that is development test and production this is done to map each of these branches to different

environments in the kubernetes cluster once you push the changes onto that particular Branch there will be a

relevant automated pipeline which will be eventually set up this means that whenever there is a change for that

specific Branch the pipeline deploys to that environment it also identifies

tests and verifies that the environment looks all right this way when a

developer makes a change in their development branch and once the development Branch succeeds they will

eventually be able to merge pull requests in order to join it to the production branch and once you click on

merge that is when it will all deploy to the production environment if you want to do a rollback you can simply create

another pull request to roll back to that particular previous state of the branch so if a user goes and changes the

code in the git repository it creates a container image and that container image is pushed to the container registry

which is eventually updated into a config updater once you create a pull request to merge to a different branch

that that is when it deploys to the concerned branch and then it tests whether these are good this way every

time you raise a pull request you know what you're merging and that pull request is being reviewed by somebody

based on the success criteria of that particular automated Branch pipeline

this is how git Ops helps teams in solving the automation problem thus githubs can be summarized as these two

things the first one is that it is a path towards a developer experience for managing applications where end-to-end

CI CD pipelines and git workflows are applied to both operations and development the second one is that it is

an operating model for kubernetes and other Cloud native Technologies providing a set of best practices that

unify deployment management and monitoring for containerized clusters and applications git Ops is based off of

four main principles so to start managing your cluster with Git Ops workflows the fall following must be in

place the first one is that the entire system described declaratively kubernetes is just one example of many

modern Cloud native tools that are declarative and that can be treated as code declarative means that

configuration is guaranteed by a set of facts instead of by a set of instructions with your applications

declarations versionites in get you have a single source of Truth your applications can then easily be deployed

and rolled back to and from kubernetes and even more importantly when disaster

strikes your cluster's infrastructure can also be dependably and quickly reproduced the second one is the

canonical desired System state versionized in git with the Declaration of your system stored in a version

control system and serving as your canonical source of Truth you have a single place from which everything is

derived and driven this trivializes rollbacks where you can use a git River to go back to your previous app

application state with gets excellent security guarantees you can also use

your SSH key to sign commits that eventually enforce strong security guarantees about the authorship and

province of your code the third one is approved changes that can be automatically applied to the system once

you have the declared State Captain kit the next step is to allow any changes to that system to be automatically applied

to your system What's significant about this is that you do not need cluster credentials to make a change to your

system with Git Ops there is a segregated environment of which the state definition lives outside this

allows you to separate what you do and how you're going to do it the fourth and the last principle is that software

agents to ensure correctness and alert on Divergence once the state of your

system is declared and kept under Version Control software agents can inform you whenever reality does not

match your expectations the use of agents also ensures that your entire system is self-healing and by

self-healing we do not just mean when nodes or pods fail those are handled by kubernetes but in a broader sense like

in the case of human error in this case software agents act as the feedback and

control Loop for your operations now I assume all of you have a fair idea of what gitops is moving on we'll discuss

how it works so how does git Ops work it's pretty complicated but let's try

and understand this in a simple way imagine you have every single piece of documentation all of your yml files and

every bit of necessary code required for kubernetes contained in a single git repository with the help of a few

automation tools anyone who is tasked with managing that kubernetes deployment can do a pull request edit the code and

then eventually issue a merge request to the git repository so once the merge request is complete the automated get

tops operator detects something has changed another automator declares the

changes operational and the change is then automatically deployed to the cluster so basically the two important

things you need to know the first one is that changes are made by a pull requests operating via pull request reduces the

number of variables which eventually transform deployments into crime scenes fortunately code deployers are no longer

awarded what hero status for manually moving bits from point A to point B the

second thing is that convergence ensures eventual consistency if a group of configuration updates is made by a human

they observe state in the cluster can drift from the desired State declared in get thus causing Divergence when the

desired and observed states are different kubernetes provides a convergence mechanism to drive the

observed State towards what the teams declared as a desired state in Version Control now let's discuss a tip local

developer workflow for creating or updating a feature a pull request for a

new feature is pushed to GitHub for review the code is reviewed and approved by a colleague so after the code is

revised and re-approved it is merged to get the git merge eventually triggers

The Continuous integration and build pipeline it runs a series of tests and

eventually builds a new image and deposits this new image to a registry the deployment automator watches the

image registry notices the image pulls the new image from the registry and updates its yml in the config repository

the deployment synchronizer which is installed to the cluster detects that the cluster is out of date it pulls the

change manifests from the config repository and deploys the new feature to production so in this workflow that I

just talked about automation is extremely crucial so as to keep everything moving with the level of

efficiency that cannot be matched with manual Administration the git Ops score

Machinery is in its CI CD tooling with the critical piece being continuous deployment that eventually supports git

cluster synchronization so now we'll move on and talk about the benefits of applying get Ops best practices the

first one is increase productivity continuous deployment automation with an integrated feedback control Loop speeds

up mean time to deployment your team can ship 30 to 100 times more changes per

day increasing overall development output by almost 2-3 times the second

reason is enhanced developer experience push code and not containers developers

can use familiar tools like get to manage updates and features to kubernetes more rapidly without having

to know the internal of kubernetes newly onboarded developers can get quickly up

to speed and be productive within days instead of months the third benefit is improve stability when you use git

workflows to manage your cluster you automatically gain a convenient audit log of all cluster changes outside of

kubernetes an order dial of who did what and when to your cluster can be used to

meet SOC 2 compilence and ensure stability the fourth benefit is higher

reliability with git's capability to revert or roll back and Fork you gain

stable and reproducible rollbacks because your entire system is described in git you also have a single source of

truth from which to recover after a meltdown reducing your meal time to recover from hours to minutes

the fifth benefit is consistency and standardization because git Ops provides

one model for making infrastructure applications and kubernetes add-on changes you have consistent end-to-end

workflows across your entire organization not only are you continuous integration and continuous deployment

pipelines all driven by pull request but your operations tasks are also fully

reproducible through git the last and most important benefit is stronger security guarantees get strong

correctness and security guarantees backed by the strong cryptography used to track and manage changes as well as

the ability to sign changes to prove authorship and origin is key to a secure

definition of the desired state of the cluster [Music]

get interview questions so before we move any further let us have a quick look at the agenda for today's session

we will start this session by The Beginner's level of interview questions then we will proceed to the intermediate

level questions and finally we will conclude with the advanced level of interview questions asked on git I hope

I'm clear with the agenda guys pretty simple so without any further delay let us begin with our first topic that is the

basic level of or the beginner level of interview questions so the first question is what is git git is a

distributed Version Control System it lets you track changes made to a file and allows you to revert back to any

particular change that you wish to it is a distributed architecture that provides many advantages over other version

control systems like the SVN one of the major advantage is that it does not rely

on a central server to store all the versions of our project's files so what happens in gitis instead every developer

clones a copy of a repository as you can see in this diagram and he has the full history of the project available on his

hard drive so whenever there is a server outage all you need to do is just copy one of your teammates locally get

repository and bang you have the code on your system so there is one Central Cloud repository where the developers

can commit changes and share them with the other team members so this is what git is in nutshell so the next question

is what is a distributed version control system so these are the systems that don't rely on a central server to store

a project file and all its versions you can see the diagram that is on your slide right now in the distributed

Version Control System every contributor can get a local copy or a clone of the

main repository here every programmer can maintain a local repository which is

actually the copy or the clone of the central repository which is present on their hard drive so developers can

commit and update the local repository without any hassles with an operation called pool they can update their local

repositories with the new data from the central server and pull operation affects changes to the main repository

from their local Repository the next question is what are the advantages of using a Version Control System

so with Version Control System all the team members are allowed to work freely on any file at any given time Version

Control System gives you the flexibility to merge all the changes into a common version

all the previous versions and variants are neatly packed up inside the version control system or the VCS you can

request any version at any time as per your requirement and you'll have a snapshot of the complete project right

at your hand so whenever you save a new version of your project your VCS requires you to provide a short

description of the changes that you have made Additionally you can see what changes are made in the files content

this helps you know what changes have been made in the project and by whom finally a distributed Version Control

System like git allows the team members to have a complete history of the project so that if there is a breakdown

in the central server you can use any of your teammates local git Repository

so these were a few advantages of using the Version Control System the next question is what is the difference

between git and SVN so git is a decentralized Version Control tool

whereas SVN is a centralized Version Control tool git belongs to the third generation of the Version Control tools

whereas SVN belongs to the second generation of the Version Control tools so what happens in git is clients can

clone entire repositories on their local systems whereas in SVN version history

is stored on the server side of the repository in git commits are possible even if you are offline whereas in SVN

only online commits are allowed so in git the push and pull operations are comparatively faster whereas in svan the

push-pull operations are comparatively slower git works are shared automatically by commit command whereas

in SVN nothing is shared automatically so these were a few differences that you can mention in this question okay so the

next question is what is the difference between get and GitHub so as we have already seen git is a version control

system of distributed nature that is used to track changes in the source code during software development so get AIDS

in coordinating work among programmers but it can be used to track changes in any set of files so the main objectives

of get are speed data Integrity support for distributed non-linear workflows

whereas GitHub is a git repository hosting service plus it adds many of its

own features so GitHub provides a web-based graphical interface it also provides access control and several

collaboration features basic task management tools for every project so the next question is what language is

used in git so here instead of just telling the name of the language you should also mention the reason for using

this language as well so git uses C language git is a fast and C language

makes this possible by reducing the overhead of run times associated with high level languages the next question

is mention the various git repository hosting functions the following are the main gate repository hosting functions

as you can see on the slide GitHub gitlab bitbucket Source force and git Enterprise and as per my opinion the

GitHub is the most popular among these options so the next question is what is a repository in git so repository in git

is a place where git stores all the files so git can store all the files either on the local repository or on the

remote Repository the next question is what is a bear repository in git so a bear repository

in git contains information about the Version Control and no working files and it doesn't contain the special dot get

subdirectory instead it contains all the contents of the dot get subdirectory directly in the main directory itself

whereas the working directory consists of two things the first one is a DOT get subdirectory with all git related

revision history of your repository and the second one is a working tree or checked out copies of all your project

files the next question is what is a commit message so a commit message is nothing but an information that you

provide along with the command called commit so that you are aware of what exactly that change the developer or the

programmer is doing while committing any operation or any change so as you can see here the Syntax for commit messages

get commit the option and the message that you want to write so the next question is explain some basic get

commands so as you can see here on this slide these are the various commands some

basic git commands that you should be aware of so the first command is the get init command as we have already seen

what is a repository in git so this command is used to create a new local repository the next command is the get

status command so this command lists the files that you have changed and those you need to add or comment the next

command is git clone so this command creates a working copy of a local repository and the syntax is get clone

and the URL the next command is the get add command so this command adds one or

more files to the staging area of git so what exactly is staging area we will see in the subsequent part of this session

the next command is get commit command as we have seen previously what commit message is that time we discussed just

one thing about the commit command so this command commits any files you have added with Git add and also commits any

files you have changed since then and the syntax is pretty much simple get commit and the option the next command

is git push origin master so this command sends changes to the master

branch of your remote Repository so these are the some basic commands that you can explain so the next

question is how do you fix a broken command so in order to fix any broken commit you can use the command git

commit hyphen hyphen amend so when you run this command you can fix the broken commit message in the editor so the next

question is what is a conflict in gate so git can handle on its own most merges

by using its automatic merging features there arises a conflict when two separate branches have made edits to the

same line in a file or when a file has been deleted in one branch but it has been edited in the other branch so

conflicts are most likely to happen when working in a team environment the next question is how do you resolve a

conflict in git so you can refer the diagram for more information about this question so there are a few steps that

you need to follow to resolve a conflict and get the first step is identify the files

that have caused the conflict in the Second Step you make the necessary changes in the files so that

the conflict does not arise again in the third step you add these files by

using the command get add and finally to commit the changed file

use the command get commit so these were the four steps that you need to follow

to resolve a conflict and get the next question is how do you revert a commit that has already been pushed and

made public so there can be two approaches to tackle this question and you must include both

the approaches in your answer because any one of them can be used depending upon the situation

so the first approach is remove or fix the bad file in a new commit and then push it to the remote repository this is

the most obvious way to fix an error once you have made the necessary changes to the file then commit it to the remote

repository using the command get commit hyphen M and the commit message in the

second approach you can create a new comment that undoes all the changes that were made in the pad commit to do this

you need to use one command called get Reverb so we will talk about get reward in the subsequent part of this session

Syntax for get revert is very simple so get revert and the name of the comment

or the commit ID the next question is what is sub git so sub git is a tool for SVN to get

migration it can create a writable get mirror of a local or a remote subversion

repository and use both subversion and get as long as you like now you can also

include some of the advantages like you can do a fast one time import from subversion to get or use sub git with

atlassian bitbucket server we can use sub git to create a bi-directional get SVN mirror of an existing subversion

Repository you can push to get or commit to subversion as per your convenience

synchronization will be done by subgit so the next question is what is the difference between get pull and get

fetch git pull command pulls new changes or commits from a particular Branch from

your central repository and updates your target branch in your local repository whereas git fetch is used for the same

purpose but it works in a slightly different way when you perform a git fetch it pulls all new comments from the

desired branch and stores it in a new branch in your local repository if you want to reflect these changes in your

target Branch git fetch must be followed with a git merge so in this case your target Branch will only be updated after

merging the target branched and the fetched Branch so just to make it very simple for you there's one equation that

you must remember so git pull equal to get fetch plus get merge so what is the

staging area or the index in git before completing the commits any change can be

formatted and reviewed in an intermediate area known as the staging area or the index so you can refer the

diagram on your slide for more reference so this will give you a clear cut idea of what exactly is the staging area the

next question is what is the use of get insta web git insta web is used to automatically direct a web browser and

run a web server with an interface into your local repository so now we will have a look at the intermediate level of

interview questions so the next question is what is get E Street so git history

represents a tree object including the mode and the name of the each item and

the sha1 value of the block or the tree the next question is what is git stash

so in this question you should first explain the need for get stash so often

when you have been working on part of your project things are in a messy State and you want to switch branches for some

time to do something else so the problem here is you don't want to do a comment of half done work just so

that you can get back to this point later so to solve this issue there is one provision called get stash so

stashing takes your working directory that is your modified track files and Stage changes and saves it on a stack of

Unfinished changes that you can reapply at any time the next question is what is

the function of get stash apply so as we have seen previously if you want to continue working where you had left your

work then get stash apply command is used to bring back these CRP changes onto your current working directory the

next question is what work is restored when the deleted branch is recovered so the files which were stashed and

saved in the stash index list will be recovered back any untracked files will

be lost also it is a good idea to always stage and commit your work or stash them

so if you want to fetch the log references of a particular Branch or a tag then run the command get reflog and

the reference name so we will see about get reflog in the subsequent part of this session okay so the next question

is explain the difference between get status and git diff so git diff depicts

the changes between the comments commit and the working tree Etc whereas get status shows you the

difference between the working directory and the index git status is helpful in understanding get a bit more

comprehensively git diff is similar to git status the only difference is that it shows the differences between the

various commits and also between the working directory and the index the next question is mention the

differences between the get clone and git remote so git remote creates an entry into your git config that

specifies a name for a particular URL whereas get clone creates a new git

repository by copying an existing one located at the URL the next question is

what is git stash drop so git stash drop command is used to remove the start item

it will remove the last added slash item by default and it can also remove a

specific item if you include it as an argument the next question is how do you

find a list of file that has changed in a particular comment so to get a list of

files that has changed in a particular comment use the command as you can see here on this slide get div tree hyphen R

and the hash so given the commit hash this will list all the files that were changed or added

in that commit The Hyphen R flag makes the command list individual files rather than collapsing

them into the root directory names only you can also include the below mention Point although it is totally optional

but this will help you impress the interviewer so the output will also include some

extra information which can be easily suppressed by including two flags so as you can see here on the screen this

command here you have included two flags the no comment ID and the name only

so here the no comment ID will suppress the commit hashes from appearing in the

output and the name only flag will only print the name of the files instead of their paths the next question is what is

the function of get config so git uses your username to associate comments with an identity the git config

command can be used to change your git configuration including your username so suppose you want to give a username and

an email ID to associate a commit with an identity so that you can know who has made a particular comment so for that

there are two commands as you can see on this slide get config hyphen hyphen Global user.name and your name that you

want to give to the user and then the email so the command for that is get

conflict Global user.email and your email address so the next question is

what does the comment object contain so commit object basically contains three components first a set of files

representing the state of a project at any given point of time the second one is the reference to the parent commit

objects and the third one is an essay one name which is a 40 character string that uniquely identifies the commit

object the next question is describe the branching strategies that you have used so this question is asked to test your

branching experience with Git so here we are going to discuss the three branches starting with the first one that is a

feature branching so a feature Branch model keeps all of the changes for a particular feature

inside of a branch when the feature is fully tested and validated by automated tests the branches then merged into the

master the second is the task branching in this model each task is implemented on its

own Branch with the task key included in the branch name it is easy to see which

code implements which task by just looking at the task key in the branch name and the third branching is the

release branching once the developed branch has acquired enough features for a release you can clone that Branch to

form a release branch creating this Branch starts the next release cycle so

no new features can be added after this point only bug fixes documentation generation and other release oriented

tasks should go in this Branch once it is ready to ship the release gets merged

into the master and tagged with a version number in addition it should be merged back into the develop branch

which may have progressed since the release was initiated so this was about the branching strategies okay so the

next question is explain the advantages of forking workflow there is a fundamental difference between the

forking workflow and the other popular get workflows rather than using a single server site to act as a central code

base it gives every developer their own server side Repository the forking workflow is commonly seen in

public open source projects a crucial advantage of forking workflow is that contributions can be generated even

without needing everybody to push to a single Central repository that leads to the clean project history developers can

push to their own server-side repositories but only the project maintenance can push to the official

repository if developers are ready to publish a local commit then they push the commit to their own public

repository and not the official one so after this they go for a pull request with the main repository that lets the

project maintainer know an update is ready to be integrated so these were a few advantages of forking workflow so

the next question is how will you know and get if a branch has already been merged into the master so the answer to

this is pretty direct so there are two commands to check this the first command is the get Branch hyphen hyphen most so

this command lists the branches that have been merged into the current branch and the second one is the git Branch

hyphen iPhone no hyphen merged so this command lists the branches that have not been merged the next question is why is

it desirable to create an additional commit rather than amending an existing comment so there are two reasons for

this question so the first one the amend operation destroys the state that was previously saved in a comment if there

is just the commit message that is being changed then that is not a problem but if the contents are being amended then

the chances of eliminating something important remains more the second reason is that abusing get hyphen hyphen amend

command can result in the growth of a small commit and acquire unrelated changes the next question is what does

hook comprise of and get this directory consists of shell scripts that are activated if you run the corresponding

git commands for example git will try to execute the post commit script after you

have run a comment the next question is in git how would you return a commit that has just been pushed and made open

so one or more commits can be reverted through the use of get reward this command in a true sense creates a new

commit with patches that cancel out the changes introduced in the specific comments if in case the commit that

needs to be reverted has already been published or changing the repository history is not an option then in such

cases gitreward command can be used to revert the commits so alternatively there is always an option to check out

the state of a particular comment from the past and committed a new the next question is how to remove a file from

get without removing it from your file system so one has to be very careful during a git add else you may end up

adding the files that you didn't want to commit however git RM command will remove it from both your staging area as

well as your file system but this may not be what you want so instead you can use the command as you can see here get

reset file or the Eco file name and Dot get ignore so this means that git reset

is exactly opposite to the get add command so the next question is could you explain the get workflow so to

record the history of the project get workflow employs two parallel long running branches the first one is the

master branch and the second one is the develop Branch so the master branch is always ready to be released on live with

everything fully tested and approved that is production ready State the hotfix branch in master is used to

quickly patch the production release these branches are a lot like the release branches and the feature

branches except that they are based on Master instead of the develop and the second branch is the develop Branch this

is the branch to which all the feature branches are merged and where all the tests are performed only when

everything's been thoroughly checked and fixed it can be merged into the master here there's one more branch called

feature Branch so each new feature should reside in its own branch which can be pushed to develop Branch as their

parent one so the next question is explain the difference between the head working tree and index so guys you can

refer the diagram that is on your slide for Clear understanding the working tree or the working

directory or the workspace is the directory of source files that you are able to see and edit

so the index or the staging area is a single large binary file which lists all

the files in the current Branch their sha1 checksums time stamps and the file

name it is not another directory which contains a copy of files within it whereas head is used to refer to the

last commit in the currently checked out branch so the next question is what is the difference between folk branch and

clone a folk is a copy of a repository normally you Fork a repository so that

you're able to freely experiment with changes without affecting the original project most commonly folks are either used to

propose changes to someone else's project or to use someone else's project as a starting point of your own idea

git cloning means pointing to an existing repository and make a copy of that repository in a new directory at

some other location the original repository can be located on the local file system or on the remote machine

accessible supported protocols the git clone command is used to create a copy

of an existing git repository in very simple word git branches are individual projects within a get Repository

different branches within a git repository can have completely different files and folders or it could have

everything the same except for some lines of code in a particular file the next question is what are the

different ways by which you can refer to a commit so in git each commit has a unique hash these hashes are used to

identify the corresponding comments in various scenarios for example while trying to check out a particular state

of a code using the git checkout hash command along with this git maintains a number of alliances to certain comments

known as the refs also every tag that is created in this repository effectively

becomes a ref and that is exactly why you can use tags instead of committing hashes in various get commands so git

also maintains a number of special analysis that are changed based on the state of the repository so in git

commits are allowed to be referred to as relative to one another in case of most comments where the

commit has two parents the raised to sign can be used to select one of the two parents however these can also be

used to refer to comments that reside on remote branches allowing one to control and manipulate them from the local get

environment the next question is what is the difference between rebasing and merging so in git the rebase command is

used to integrate changes from one branch into another it is an alternative to the merge command the difference

between rebasing and merging is that rebase rewrites the commit history in order to produce a straight linear

succession of commits merging is gets way of putting a forked history back together again the get merge command

helps you take the independent lines of development created by get Brands and integrate them into a single Branch the

next question is mention the difference between revert and reset so reset is used to return the entire working tree

to the last committed state this will scrap comets in a private Branch or throw away uncommitted changes reset

changes which commit a branch head is currently pointing at it Alters the existing comment history so this command

can also be used to unstage a file whereas get reward command creates a new comment that undergoes the changes from

a previous commit so this command adds new history to the project it doesn't modify the existing history now we will

talk about the advanced level of interview questions so the question here is how to squash last n comments into a single comment so

there are two options to squash last and comments into a single commit and you should mention both of these options

while answering such questions so in the first option if you want to write the new commit message from scratch you

should use the following command as you can see here on the screen git reset soft head and git commit and in the

second option if you want to start editing the new comment message with a concatenation of the existing commit

messages then you need to extract those messages and pass them to get commit and for this purpose you will use the

command get reset soft head and this command get commit edit hyphen M and the

entire lock command get log format reverse and the name of the head the

next question is what is get bisect how can you use it to determine the source of a bug so git bisect is used to find

the commit that introduced a bug by using a binary search so the command for git bicep is pretty simple as you you

can see on this slide get bisect sub command and the options this command uses a binary search algorithm to find

out which commit in your Project's history introduced up bug you can use it by first telling it a bad commit that is

known to contain a bug and a good comment that is known to be before the bug was introduced then get bisect picks

the commit between these two endpoints and asks you whether the selected comment is good or bad so it continues

narrowing down the range until it finds the exact commit that introduced the change so the next question is how do

you integrate git with Jenkins so as you can see on this slide these are the four steps that you can follow to integrate

git with Jenkins and I am also going to show you the screenshots for these four steps so first anyone who is aware of

Jenkins or has used Jenkins you must be aware of this home page or the Jenkins dashboard so here you need to click on

manage Jenkins and once you click on manage Jenkins you will find this option here the manage plugins option click on

that and then and you will have to search for the plugin here as you can see in this screenshot click on this

plugin and here you have these two options installed without restart download now and install after restart so you can choose any one of them and

once you are done with that here you will be able to see this plugin that we have just downloaded along with its

version number so these are the steps that you must follow to integrate get with Jenkins so the next question is how

do you configure a repository to run code sanity checking tools right before making comments

so here you must first mention what exactly is sanitary checking a sanity or smoke test determines whether it is

possible and reasonable to continue testing and now you must explain how do you achieve this so this can be done

with a simple script related to the pre-commit hook of the Repository the pre-commit hook is triggered right

before a commit is made even before you are required to enter a commit message so in this script one can run other

tools such as lenders and perform sanity checks on the changes being committed into the Repository

so as you can see here this is the script that I'm talking about the purpose of the script is that it checks

to see if any dot go file that is about to be committed needs to be passed through the standard go source code

formatting tool by exiting with a non-zero status the script effectively prevents the commit from being applied

to the repository the next question is what is get cherry pick the command gate

cherry pick is normally used to introduce a particular comments from one branch within a repository onto a

different branch another common use of this command is to forward or backboard commits from a maintenance Branch to a

development Branch this is in contrast with the other ways such as merge and rebase which normally apply many

comments onto the another Branch so the Syntax for this command is very simple you just need to type git space Cherry

hyphen pick space the commit hash the next question is what is get reflog the

reflog command keeps a track of every single change made in the references that is the branches of the tax of a

repository and keeps a log history of the branches and the tax that were either created locally or checked out

reference logs such as the commit snapshot of when the branch was created or cloned checked out renamed or any

comments made on the branch are maintained by get and listed by the reflog command so this command must be

executed in the repository that has the Lost branch so if you consider the remote repository

situation then you have to execute the reflock command on the developer's machine who had the branch so the next

question is how to recover a deleted Branch using the getreflow command so here as you can see on the screen these

are the three steps that you must follow to recover a deleted branch and I'm also going to show you the screenshots okay

so in the first step you get the history logs of all the references for that you just need to type get reflog and once

you type this command you can see the history log in the Second Step you need to identify the timestamp of the branch

that you want to recover so here in this case this is the timestamp that we are looking for the pre plot Branch here in

the third step you just need to use the command get checkout hyphen B and the branch name this is a pointer reference

when the pre-prod branch was created so if you follow these three steps you will be able to recover the deleted Branch so

the last question of this session that is mention some of the git reflux sub commands so as you can see on the screen

these are the commands that we will talk about four to five commands the first command is git reflog hell so this

command is used to open up the manual page the second command is the get reflock show so this command shows the

logs of the references provided in the command line the third command is the get reflock expire so this command is

used to prune the older reflog entries the fourth command is the git reflock

delete so as the name suggests this command deletes single entries from the reflog history and the fifth command is

get reflock exists so this command checks whether a reference that is a branch or attack has a reflux log

history entries so these were a few commands like get reflock sub commands that you can answer in this question so

with this we have come to the end of this session I hope you have enjoyed this session if you have any queries you

can post them in the comment box below and we will try to get back to you as early as possible till then Happy

learning thank you I hope you have enjoyed listening to this video please be kind enough to like

it and you can comment any of your doubts and queries and we will reply them at the earliest do look out for

more videos in our playlist And subscribe to edureka channel to learn more happy learning