# GIT AND GITHUB

# Git

## Local repository(everybody in the team must maintain local repo along with workspace)

## It is a tool.

# GitHub

## Remote repository

## It is a service to use git.

## First the changes should be part of local repo and then u must send to remote repo from remote repo Jenkins will take the code using project URL then Jenkins will execute the automation. This is everyday activity.

# Prerequisite:

# Step1:

## Have account in github.

# Step 2:

## Create new repository in github.

## Give repository name (it should be your project name)->select create repository.

## You will get the repository URL(github url).

# Step 3:

## Install git software and create local git repository and add environmental variable.

# Flow:

## Git and workspace will be present in our local system.

## From workplace send the files to staging area.

## Once files are available in staging, we can send then to git repo.

## Once files are available in git, we can send then to GitHub repo.

## In each stage we will use certain number of commands

# Commands

# git init

## Create new empty git repo locally (unless and until u didn’t specify this command your project will not part of version control system)

# git remote add origin “specify remote URL”

## Connect local repo to remote repo.

## Above 2 steps will be done only once.

# git status:

## Files are ready to send to staging area(it wil show wat all changes we have done)

## ie) Add new file, modify existing file and remove the old file

## **tracked files** : part of staging and git repo.

## **untracked files**: not part of staging and git repo.

## Before sending to staging every time we have to execute status command

# git add:

## From workspace to stage area, we must use add command to add those files to staging area.

# git add -A:

## Send all files to staging area(warnings don’t need to consider only errors u consider)

# git add -specific file:

## Only specific file

# Before Doing commit first time we need to execute these below command

## git config –global user.name “priya”

## git config –global user.email “githubemailid”

# git commit -m “Write some comments”:

## Now files will be available in staging area, so we have to send it to git repository for that we use commit command

# git commit -a -m “comments”:

## add all files to staging and local repo wil be done using this single command.

# git push -u origin master

## Send all code to github repo from git repo.

## After entering push command, it will ask our github URL along with Username and Password

## Jenkins will take URL (ie we have to configure URL in jenkins) from clone and download button.

# Now some people also pushed their code to github but that will still not be the part of our local repo and workspace how to get that

# git pull origin master

## Pulls the files from github repo to git repo .

## Now in Jenkins we must create project and under source code management select Git option->Provide remote repo(github) URL.

## Under build section just give clean install-> apply and save

## Manage Jenkins->Global tool configuration->Provide complete Git path till bin.

## Wherever Jenkins is running in that machine git should be installed

# git log

## It will show how many commits u have done earlier, date and time, comments of the commit.

# git diff

## Compare files(file content) in version control system (To compare the file content between staging and local repositories or work directory and staging)

# Req 1:

## To see difference in file content between working directory and staging area

# git diff filename.txt

## a/index.txt – represents source(staging area)

## b/index.txt – represents destination(working directory)

## hash value of file from stage

## hash value of file from dir

## 3 rd hash value is git file mode(100 -represents type of file(txt,csv etc), 644- file permission)

## (4+2)6 represents – read and write

## 4 represents read

## 2 represent write

## 1 represent execute

## If anyline prefixed with space then file is unchanged

## If anyline prefixed with + means then it is added in destination copy

## If anyline prefixed with - means then it is removed in destination copy

# Req2:

## To see difference in file content between working directory and last commit

## New/latest or last commit is always represented with HEAD.

# git diff HEAD filename.txt

# Req3:

## To see difference in file content between staged copy and last commit

# git diff – staged HEAD filename.txt

# git diff – cached HEAD filename.txt

## If we didn’t give filename it will take multiple files

# Req4:

## To see difference in file content between specific commit copy and working directory

## For that we need to know the commit id so execute one command to get commit id

# git log –oneline

## This will give u details of all commit u did along with commit id

# git diff commitid filename.txt

# Req5:

## To see difference in file content between specific commit copy and staged copy

## For that we need to know the commit id so execute one command

# git log –oneline

## This will give u details of all commit u did along with commit id.

# git diff --staged commitid

# Req6:

## To see difference in file content between two specific commit.

# git diff commitid1 commitid2 filename

# Req7:

## To see difference in file content between 2 branches

# git diff master(branch1) test(branch2)

# Req8:

## To see difference in file content between local and remote repo

# git diff master(local branch) origin(URL of remote repo)/master(branch)

# git rm

## Remove/delete files from version control systems.

# Req1:

## Remove file from both staging and working directory.

# git rm filename.txt

# ls

## check files in working directory.

# git ls-files

## check files in staged area.

# Req2:

## Remove all files from both staging and working directory.

# git rm -r .

# Req3:

## Remove files only from staging not in working directory.

# git rm --cached filename.txt

# Req4:

## Remove files only from working directory.

# rm filename.txt

# git checkout

## Undo or discard operation

## The file is having some content or changes that is not part of staged area or local repo but that File is already part of staged area(only new changes not part of staged or local repo but Initially during creation it is added to staged area)

## This will work only for working directory.

# git checkout filename.txt

# git reset

## Remove changes from staging area.

# git reset filename.txt

## Undo commits at repo level . for this we must follow some syntax and we need commit id

# git reset <mode> <commit id>

## Moves the head to the specified commit, and all remaining recent commits will be removed.

## Mode will decide whether these changes are going to remove from staging area and work directory or not.

# Mixed

## Even we didn’t mention mode in command it is default mode and discard commit in both staging and local repos.

## It wil not touch working directory.

## We can revert using add and commit

# Git reset –mixed <previous commit id or HEAD tilt symbol one>

# (which commit u want to reflect in repo put that commit id in reset so whatever is committed above that will be removed)

## Whatever commit id we are mentioned in reset , so above that all latest commit will be removed

# Soft

## Same as mixed but changes are available in the working directory and staging area.

## We can revert using git commit.

# Hard

## Same as mixed but changes will be removed from everywhere(permanently remove files)

## Only group of commit ids will be discarded.

## No way to revert.

## Reset is used only when your commits are in local repo. If used in remote, then there will be conflicts and chance of losing data.

# git revert:

## Changes are all removed but it done extra reverse commit. and record some new commits that record them.

## Appends a new commit with the resulting inverse content

## This revert commit is used when u know that all your code and commit went into your remote and also all developers has pulled your data so that all commits has been existing in local system.

## safe method

# git stash:

## If you want to switch to branch 2 but you have some work already done in branch 1 in that case, git will not allow you to switch to other branch without committing the modified code.

## But we don’t want to commit the half done work.

## So now git stash(stores something safely) command help us to switch to other branch(branch 2) without committing the modified changes in branch1.

## Now Git temporarily saves the data safely without committing.

# git stash list

## It will give you list of stash u done(u can check whether stash is occurred or not).

# git stash pop

## It will get back the changes from the stash and it will remove it in the stashes and it will give it to the branches.

# git stash save <stash name>:

## To give custom name for the stash u are about to save before switching to other branch.

# git stash apply

## It will get back the changes and give it to the branches from the stash without removing changes from stashes.

# ****git stash drop stash@{1}****

To delete any particular stash (For ex:– stash@{1}), use ‘git stash drop stash@{1}’. By default, this command will drop stash@{0} if no argument is provided (git stash drop).

# ****git stash clear’****

## To delete all stashes at once, use the **‘****git stash clear’** command.

# Branching:

## As soon as we create repository and first commit master branch(default branch) will be created so whatever file u r staging and committing it wil be moved to master branch .

## Master is main branch(source code will be in master branch)

## Without disturbing main branch, we can create another branch so not to corrupt the existing code.

## Finally, he can push the code into remote repository, or he can merge into master branch.

## Main usage of branching is parallel coding, and we can have clean code without disturbing main source code we can create multiple branches and we can implement whatever feature.

## After completion of the feature, we can Finally push the code into remote repository, or he can merge into master branch.

## Every branch is independent and isolated whatever files we are adding and changing that will not be present in master and will not impact master branch.

# Git branching commands:

# git branch:

## To view available branches

## \* - represents active branch (ie currently we are working on master)

## git status command also show u the available branches.

# git branch branchname

## To create new branch

# git checkout branchname

## Switch from one branch to another branch

# git checkout -b branchname

## Creating and switching to same branch

# Merging:

## When I do parallel development when we complete the feature branch that code will be combined into master branch

# Two ways we do merging

# Fast forward merge

## we do changes only in the child branches after created it from parent branch , we don’t do any changes in the parent branch(master).

## So, there will be no chance to have conflicts.

# Three-way merge(recursive strategy)

## Parallelly changes are happening in the master branch as well as child branch.

## So, there will be many chances to have conflicts.

## After merging it will create a new commit called merge commit

## Git will create this merge commit.

# Merging

## When u create new branch and u want to combine or merge with master branch the command always executed from master branch

# git merge branchname(childbranch)

## During three-way merge it will open one editor and show new message and we have to save the file with :wq!(save and exit)

## If we do some modifications in file there will be conflict

# How to resolve conflicts:

## We must resolve merge conflicts manually.

## Open the file it will show u clearly the conflict.

## Keep the line u want and remove the remaining line,

## Save the file.

# Using git merge tools

## After resolving conflict, we have again modified file and committed the file merge into master branch git will create merge commit.

## To view it in graphical format

# git log –oneline –graph

## If all changes are moved to master branch from child/feature branch, then we can remove feature branch

# git branch -d branchname

## History is maintained in merge.

## Linear- merge commit have two parents(master and child)

# REBASE(alternative of merge)

## Internally follows fast forward merge.

## In rebase no merge commit will create

## When we execute rebase command then it will create objects for child branch and all details of child branch will be same except commit id

## Real child branch commit will be discarded.

## Child branch objects will be added on top of master branch last commit.

# Steps:

# git rebase master

## Adding last child branch commit on top of last master branch commit. In that case it will create two objects from the child commit and the main child commit will be discarded but then objects will contain same details except commit id details.

## We should switch to child branch and must execute above command.

## After rebase master branch last commit will become base commit and child branch last commit will become main commit (head)(added on top of master)

## Again, it becomes fast forward merge.

# git merge childbranch

## Rebase should be used only within local environment not for remote repo(not recommended)

## In rebase it is very difficult to identify who is doing what if in local then we ourself create multiple branches and rebase ourself

## Non-linear – every commit has one parent.

# git pull

## We already have repo locally but there are some new changes in my remote repo so it will pull all the new changes from remote repo to local repo.

# git push

## whatever code we have done in local repo push that code into remote repo

# git clone

## Create local repo based on the remote repo(create copy of remote repo into local repo

# git fetch

## Only let us know the changes it will not try to merge those changes into our local repo.

**$ git merge [branch] --strategy-option ours**

## In my case, I wanted to accept all current changes and ignore any incoming changes, which I could accomplish like this.

**$ git merge [branch] --strategy-option theirs**

## If, instead, you know you want to overwrite any current changes and accept all conflicts from incoming changes, you can use the theirs strategy instead.

# To create file directly in github

## Click repo name ->click create a new file -> give name for file -> add any code->click commit file.

## To clone that above project to local repo

# git clone ‘Github URL’

## To push changes done in local to remote repo

# git push origin main(master)

## To pull changes from remote repo

# git pull origin main(master)

# Clone using eclipse IDE:

## Right click->import->Git->Projects from git->Next->Clone URI->Next->copy and paste the remote repo URL in URI field->next->next->next->Finish.

# Once I did the changes and modifications then I want to push it to the local repository so,

## Right click on file-> Team-> add to index.

## Right click on file-> Team-> commit(add comments)

## Also provide credentials of your Github.