###############

19th Dec. 2023

###############

Module - 3

Version Control System using Git!

Version Control System!

Project!

Web\_Application!

index.html #

----

---

--

---

---

-----

----

save index.html

Open a web browser & test this index.html page!

Next DAy !

index.html

----

---

-asdfs-

---

-adfs--

-----asdfas

----asdfasdfa

asdfasdfasd

factasdf

asdfsad

fs

aboutsdf

save index.html

Open a web browser & test this index.html page!

Version Control the Changes!

index.html

If we save the same file using Version Control System !

index.html\_v1.0 # v1.0 - tag/version\_Number/commit\_id

index.html\_v1.1

index.html\_v1.2

index.html\_v1.3

index.html\_v1.4

index.html\_v1.5 # Latest version!

Version Control System :

- Used to version control the source code changes

- Used to Track the Code Changes

- Used to quickly indentify the previous changes and revert.

GIT is one of the Open Source Distributed VCS.

Types of Version Control System :::

- Local VCS

- Centralized VCS

Repository -- collection of folders and files - usually we maintain the project source code.

- Users should always be online to work in the project

- If the Central server get currpted/losts, we lose the entire data/project.

- Distributed VCS

- The Remote server maintains the source code repository.

- Backup of Remote Repositories

- Source Code Repository is distributed in all the machines.

- Users can have their local copy of the repository and work offline

GIT is one of the Open Source Distributed VCS.

Install GIT Client in the local machine.

GITHUB is considered as one of the Remote GIT Repository Servers

Azure Repository

bit bucket

gitlab

AWS code commit

git fork ????

- It copy one remote repository to an another remote repository.

What is

- GIT is one of the Open Source Distributed VCS.

- Used to version control the source code changes

- Used to Track the Code Changes

- Perform Parallel Development

Next ::

GIT

Install GIT Client in Local Machine

Git Commands

###############

21st Dec. 2023

###############

GIT

Install GIT Client in Local Machine

Git Commands

Distributed VCS :::

Local Machine ====> Remote Server

Git Client (installed in Local Machine) GITHUB

Repository :

Repository -- collection of folders and files - usually we maintain the project source code.

Git File Workflow :::

Local Machine ====> Remote Server

Working Directory -----> Staging Area --------> Local Repository -------> Remote Repository

index.html git add index.html git commit index.html\_v1.0 git push index.html\_v1.0

Developers' Workload ???

- New Project

- Enhancement Project / Bug fixing!

GIT CLI Commands :::

git clone --> Used to clone/copy the entire Remote Repository into the local Machine

git add --> Used to add the changes from Working Directory to Staging Area

git commit --> Used to commit the changes from Staging area to Local Repository

git push --> Used to push the changes from Local Repository to Remote Repository

git fetch/git pull ::

--> Both git fetch and git pull are used to handle the incremental changes from Remote Repository

--> git fetch :

It is just to check for any incremental changes in remote repository.

If there is any incremental changes exist in the remote repository, it will update the changes in the Local Repository. It will never Update the Changes in Working Directory.

--> git pull :

It will check for any incremental changes in remote repository.

If there is any incremental changes exist in the remote repository, it will update the changes in the Local Repository as well as in the Working Directory.

fork --> Used to copy a Remote Repository to another Remote Repository

Git Client (installed in Local Machine)

https://git-scm.com/downloads

Choose windows select standalone installer - 64bit

Working with GIT CLI :

Maintain the Project/Workspace :

Folder Structure!

cd d:

d/DevOpsGIT

cd d:

mkdir DevOpsGIT

cd DevOpsGIT

pwd

# d/DevOpsGIT

# c/DevOpsGIT

git --version

mkdir project-repo1

git init --> Is used to Initialize a new git Repository

It creates a GIT\_DIR! and the default branch (master/main)

git status

git config --global user.email "loksai@asdfasd.com"

git config --global user.name "LoksaiETA"

echo "rec1" >> file.txt

git add file1.txt

git status

git commit -m "Created File1.txt"

git status

git status

git log # get the list of commit we made in the repo.

mkdir project-repo2

git init --> Is used to Initialize a new git Repository

It creates a GIT\_DIR! and the default branch (master/main)

git status

git config --global user.email "loksai@asdfasd.com"

git config --global user.name "LoksaiETA"

echo "rec1" >> file.txt

git add file1.txt

git status

git commit -m "Created File1.txt"

git status

git status

git log # get the list of commit we made in the repo.

###############

22nd Dec. 2023

###############

GIT CLI Commands

Working with Local GIT Repository!

create

git add - Used to add the changes from Working Directory to Staging Area

git add <file\_name>

git add <file1.txt> <file2.txt> <file3.txt>

git add \*.txt

git add \*.java \*.md

git add . # this is most commonly used

git commit -m "Commit Message" # it is important to enter valid commit message.

Release\_Number

Change Request Number

git commit -m "Created project files"

git status # get the current status of the repository.

git log # list the commits and used for tracking the changes.

git log

git log --oneline

git log -2

git log --oneline -3

git show <commit\_id>

ls # Is a Linux command used to get the list of files in working directory/untracked files.

git ls-files # used to get the list of files that are being tracked by git.

git rm --cached file1.txt # used to remove the file Just from staging area.

git rm -f s1.txt # used to remove the file from the staging area as well as from working directory.

HEAD Pointer # Is a pointer, used to point the latest commit of the current Branch.

Undo the Commited Changes from Repository ::

git reset

syntax:

git reset <reset\_option> <previous\_commit\_id>

- git reset is used to undo the committed changes from the Repository.

- git reset will never create any new commit point. - It is NOT Recommended in the Shared Repository. Because, the reset activities cannot be tracked.

- git reset options :

git reset --soft ::: It will remove the changes from local repository, changes will be moved back to Staging Area.

The HEAD Pointer will points to that specific Commit point.

git reset –mixed # This is the DEFAULT Option

It will remove the the changes from local repository as well as from the staging area, and directly move the changes back to Working Directory.

The HEAD Pointer will points to that specific Commit point.

git reset --hard # Should be AVOIDED. It will remove the the changes from local repository, staging area as well as from the Working Directory. It is a permanent Deletion of file.

The HEAD Pointer will points to that specific Commit point.

git reset --soft cm2

git reset cm2

cm3

cm2 HEAD

cm1

###############

28th Dec. 2023

###############

GIT Misc. Commands :::

git reset

git revert !

git branching

git merge and its options

git revert :::

syntax:

git revert <previous\_commit\_id>

- git revert is used to undo the committed changes from the Repository.

- git revert will create any new commit point.

- git revert is Recommended in the Shared Repository. Because, the revert creates teh commit used to tracking the changes.

- git revert is same as git reset --hard. Only difference is, we cannot track the git reset actions, but we can track the git revert action.

git branch :::

- GIT SCM used to perform parallel development.

- git branches are logical copy of repository.

GIT Branching Strategies/Techniques!

Default Branch - Master is considered as a production copy of the source code.

Master Branch is called as long live Branch

Scenario 1 ::

Repository :

master branch -- index.html -- created 10 commits - and it is working as expected!

Scenario 2 ::

Repository :

master -- index.html -- created 10 commits - and it is working as expected!

feature -- index.html -- created 10 commits - and it is working as expected!

(Short Live Branch) ++10 commits & test

Enhancement Project!

Repository :

master -- index.html -- created 10 commits - and it is working as expected in prod.

feature

feature1cm1

Repository :

master - cm1,cm2,cm3 index.html

feature1 - cm1,cm2,cm3 index.html

feature2 - cm1,cm2,cm3 index.html

feature3 - cm1,cm2,cm3

Scenario 3 ::

Repository :

master - cm1,cm2,cm3

Developer\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2 index.html

feature2 - cm1,cm2,cm3,f2cm1,f2cm2 index.html

Scenario 4 ::

Repository :

master - cm1,cm2,cm3

Integration\_Branch : - cm1,cm2,cm3,Developer1\_Changes,Developer2\_Changes,Developer3\_Changes

Developer1\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2

feature2 - cm1,cm2,cm3,f2cm1,f2cm2

Developer2\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2,f3cm1,f3cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2

feature2 - cm1,cm2,cm3,f2cm1,f2cm2

feature3 - cm1,cm2,cm3,f3cm1,f3cm2

Developer3\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2

feature2 - cm1,cm2,cm3,f2cm1,f2cm2

Scenario 5 ::

Repository :

main/master - cm1,cm2,cm3,Changes\_from\_Team1&Team2

Release\_Branch - cm1,cm2,cm3,Team1\_Changes,Team2\_Changes

Integration\_Branch1 : - cm1,cm2,cm3,Developer1\_Changes,Developer2\_Changes,Developer3\_Changes --> Team1

Developer1\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2

feature2 - cm1,cm2,cm3,f2cm1,f2cm2

Developer2\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2,f3cm1,f3cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2

feature2 - cm1,cm2,cm3,f2cm1,f2cm2

feature3 - cm1,cm2,cm3,f3cm1,f3cm2

Developer3\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2

feature2 - cm1,cm2,cm3,f2cm1,f2cm2

Integration\_Branch2 : - cm1,cm2,cm3,Developer1\_Changes,Developer2\_Changes,Developer3\_Changes --> Team2

Developer1\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2

feature2 - cm1,cm2,cm3,f2cm1,f2cm2

Developer2\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2,f3cm1,f3cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2

feature2 - cm1,cm2,cm3,f2cm1,f2cm2

feature3 - cm1,cm2,cm3,f3cm1,f3cm2

Developer3\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2

feature2 - cm1,cm2,cm3,f2cm1,f2cm2

Scenario 6 : BugFix/Hotfix Changes.

Repository :

main/master - cm1,cm2,cm3,Changes\_from\_Team1&Team2

hotfix\_branch # It is created dynamically during the Production Issues.

Next Working with Branches ::::

git commands ::

git branch

git switch -c <new\_branch\_name>

repository:

master : -- cm1,cm2,cm3

feature1 -- cm1,cm2,cm3

feature2 -- cm1,cm2,cm3

feature3 -- cm1,cm2,cm3

git branch # to get the list of branches in the repository & identify the current Branch

git switch -c branch

git branch <branch\_name>

git checkout -b <branch\_name>

git Merge :::

git merge command is used to merge the changes from the current branch to the target branch.

repository

master - cm1,cm2,cm3 # Default Branch

feature1 - cm1,cm2,cm3

git merge <source\_branch\_name> # This command should be executed from the Target Branch.

git switch master # Switch to Master Branch

git merge feature1 # Merge the Feature1 Changes into Master Branch

master :

feature4 - ready with the changes updated the same file

feature5 - ready with the changes updated the same file

Merge Conflict :::

1. How the Merge conflicts Occurs ???

- When more than one user/feature try to update the same file at the same record level, Merge Conflict will Occur.

2. How to Resolve the Merge Conflict ???

- Identify the file causing Merge Conflict.

- Review the Content of the file in target branch

- Decide which changes has to be retained.

- Remove the header/footer and unwanted records from the file.

- Add and Commit the changes in target branch.

In DevOps Practise, We should NOT Resolve Merge Conflict.

We should always Prevent Merge Conflict.

How to Prevent Merge Conflict ???

Strictly follow DevOps Practise!

###############

29th Dec. 2023

###############

rebase

squash

stash

gitignore

remote Repository -- github!

Next : Jenkins!

git rebase :::

Rebase is used to sync the latest changes from Target Branch with the current branch.

It helps maintain the linear commit history

It helps to prevent merge conflicts in the target branch.

It is one of the best practises while working in git branches.

create repo.

repo1

master - cm1

cm1,f1cm1,f1cm2 # upon merging feature1

cm1,f2cm1,f2cm2,f1cm1,f1cm2 # upon merging feature2 without Rebase.

cm1,f1cm1,f1cm2,f2cm1,f2cm2 # upon merging feature2 with Rebase.

feature1 - cm1,f1cm1,f1cm2

git switch master

git merge feature1

feature2 - cm1,f2cm1,f2cm2

git rebase master

- cm1,f1cm1,f1cm2,f2cm1,f2cm2

git switch master

git merge feature2

GIT Squash :::

Git Squash is a Merge option.

It is used to combine the more than one commits into a single commit point, It is used while merging the changes to target Branch.

If we dont want to highlight the commit history to the target Branch we use git Squash.

Repo1 :

master - cm1

cm1,f1cm1,f1cm2,f1cm3,f1cm4,f1cm5,f1cm6,f1cm7,f1cm8,..........,n #Upon Merge without squash.

cm1,Feature1\_Changes #Upon Merge with squash.

feature1 - cm1,f1cm1,f1cm2,f1cm3,f1cm4,f1cm5,f1cm6,f1cm7,f1cm8,..........,n

git rebase master

git switch master

git merge --squash feature1

git commit -m "Combined commits from feature1"

Enter the Message : Feature1\_Changes

Scenario 5 ::

Repository :

main/master - cm1,cm2,cm3,cm4 # Upon Squash!!!!

Release\_Branch - cm1,cm2,cm3,Team1\_Changes,Team2\_Changes

Integration\_Branch1 : - cm1,cm2,cm3,Developer1\_Changes,Developer2\_Changes,Developer3\_Changes --> Team1

Developer1\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2 feature2 - cm1,cm2,cm3,f2cm1,f2cm2

Developer2\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2,f3cm1,f3cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2 feature2 - cm1,cm2,cm3,f2cm1,f2cm2

feature3 - cm1,cm2,cm3,f3cm1,f3cm2

Developer3\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2 feature2 - cm1,cm2,cm3,f2cm1,f2cm2

Integration\_Branch2 : - cm1,cm2,cm3,Developer1\_Changes,Developer2\_Changes,Developer3\_Changes --> Team2

Developer1\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2 feature2 - cm1,cm2,cm3,f2cm1,f2cm2

Developer2\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2,f3cm1,f3cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2 feature2 - cm1,cm2,cm3,f2cm1,f2cm2

feature3 - cm1,cm2,cm3,f3cm1,f3cm2

Developer3\_Branch: - cm1,cm2,cm3,f1cm1,f1cm2,f2cm1,f2cm2

feature1 - cm1,cm2,cm3,f1cm1,f1cm2

feature2 - cm1,cm2,cm3,f2cm1,f2cm2

git cherry-pick :: it is used to merge a specific commit to the target branch.

It is not a best practise to use.

git cherry-pick <commit\_id>

git cherry-pick Team1\_Changes

stash::: Stash is used to store the uncommited changes to the temporary Area.(GIT\_DIR!)

Git Stash ::: Its allows to switch to any branch without commiting the changes in the current branch.

git stash save "Save\_Massage" # USed to Stash the uncommitted changes to temporary Area.

git stash list # Used to get the list of Stashed Entries

git stash apply # Apply the latest entry from the stash list back to the staging area!

# It will never remove that entry from stash list.

git stash apply stash@{1}

git stash pop # Apply the latest entry from the stash list back to the staging area!

# It will remove that entry from stash list. git stash pop stash@{1}

git stash drop # used to Drop/Delete the latest entry from the stash list.

git stash drop stash@{1}

git stash clear # used to clean-up the stash list.

.gitignore ::::

It is used to ignore the files from tracking!

It is always a best practise to have .gitignore file in the repository.

SCM :

Local Machine :

Java\_Web\_App\_Project\_Repo ===> Remote Github Repo!

src

main

test

target

\*.war

pom.xml

application.properties

db\_userID

db\_Password

db\_Secrets

db\_Credential

.gitignore

\*.class

\*.war

\*.doc

sample.txt

target/

db\_userID

db\_Password

db\_Secrets

db\_Credential

Handling Remote Repository -- github!

Developers' Workload :::

New Project

Enhancement/BugFix Projects

GITHUB

Bitbucket

Azure Repos

AWS Code Commits

gitlab

Pull Request ::: It is created by the developers and assigned to the Reviewers for review and approve.

Task :::

Practise the GIT Commands till date.

Install Jenkins in AWS VMs and Login to Jenkins using Web Browser

Next :::

Remote Repository Handling

Jenkins Module.

###############

2nd Jan. 2024

###############

Remote Repository Handling

- Local and Remote Repositories

Handling Remote Repository -- github!

Developers' Workload :::

New Project

Enhancement/BugFix Projects

GITHUB

Bitbucket

Azure Repos

AWS Code Commits

gitlab

GIT CLI Commands :::

git clone --> Used to clone/copy the entire Remote Repository into the local Machine

git add --> Used to add the changes from Working Directory to Staging Area

git commit --> Used to commit the changes from Staging area to Local Repository

git push --> Used to push the changes from Local Repository to Remote Repository

git fetch/git pull ::

--> Both git fetch and git pull are used to handle the incremental changes from Remote Repository

--> git fetch : It is just to check for any incremental changes in remote repository.

If there is any incremental changes exist in the remote repository, it will update the changes in the Local Repository. It will never Update the Changes in Working Directory.

--> git pull : It will check for any incremental changes in remote repository.

If there is any incremental changes exist in the remote repository, it will update the changes in the Local Repository as well as in the Working Directory.

git pull ==> git fetch + git Merge

fork --> Used to copy a Remote Repository to another Remote Repository

git clone :

git clone https://github.com/SA-DCP-Nov16/TestRepo1.git

github LoginID : loksaieta

github Access Token : ghp\_xExHb4U8kBZod9YvxjViGl202aur1j4M19dx

How to create github access token ?

ghp\_esQIeQ22aHv2Adxthz6xo1A8UzAlMR4HRIug

git remote

git remote -v # Used to get the list of the remote repositories linked to the local repository

git push

git push -u origin local-feature1

git fetch

git pull

git branch -M main # Rename the Branch

As a best Practice :

- Always used git rebase before merge between branches

- Always use git pull before git push.

git branch -M master # Rename Branch!!!

git remote add origin https://github.com/SA-DCP-Nov16/TestRepo2.git # Connect/Link to remote repository

git remote remove origin #Disconnect the remote repository from local

git switch -c feature1

git switch master

Summary ::

- git file workflow

- git Misc. Commands

- git branching Strategies/Techniques

- Merge

- Merge Conflict

- Rebase / squash /stash

- Remote Repo Handling!