OPERATIVE

GIT & GITHUB TRAINING

OPERATIVE

2023

Version Control System Introduction

- Definition: System to maintained several versions of files.

Need:

1. Every version is important and we need a specific version at any point of time. So its difficult to find a specific version
2. When a software is composed of various modules and each module is developed by specific employee then its difficult to find out which module is modified by which employee.
3. Need effective system to share file among multiple employee

Version Control System Types

- Types:
- 1: Centralized VCS.
- 2: Distributed VCS.

Centralized:

1. Everything is stored on server and nothing on employee's local machine
2. If server crash everything will be lost
3. Also when there are too many employee then its difficult to get specific file.

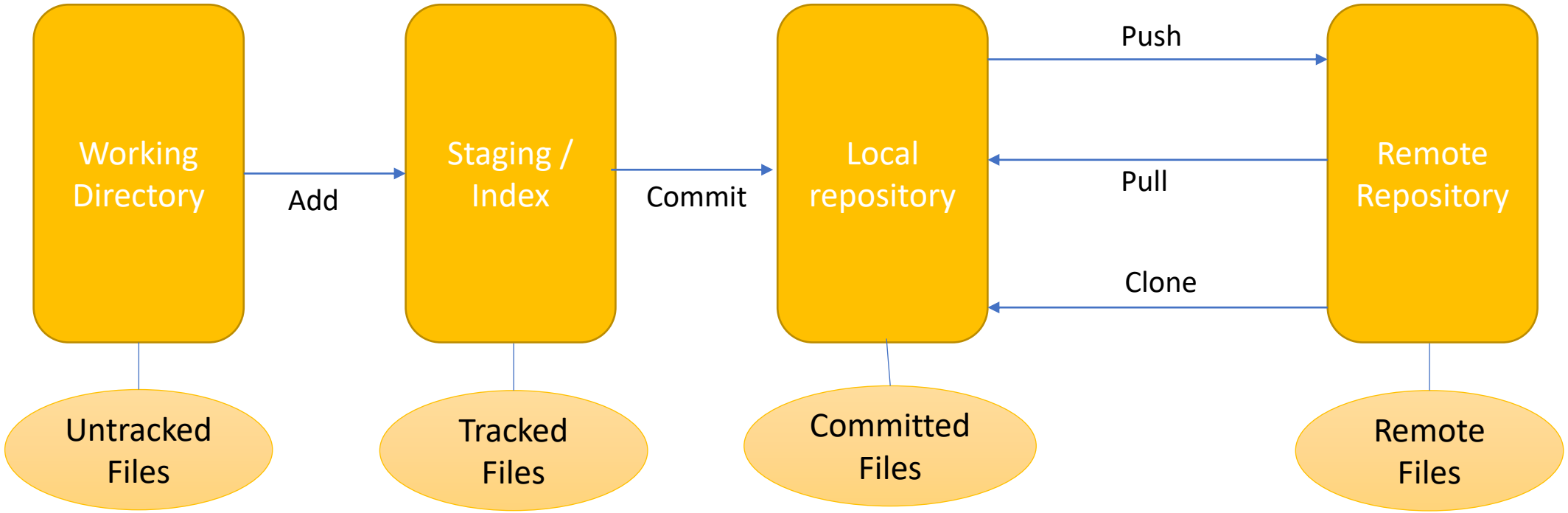
Distributed:

1. Only final copy is stored on server.
2. All versions are stored on individual's local machine

Version Control System Terminologies

Sr No	Terminology	Meaning
1	Writing Copy	Version that is stored on individual's machine
2	Work Space / Working Directory	Directory / folder created on individual's machine
3	Local Repository	Place where final copy is stored on individual's machine
4	Remote Repository	Place where final copy is stored on server and shared with employee
5	Checkout, Commit, Push, Pull, Git, GitHub	We need to understand its architecture first

Version Control System Architecture



Version Control System Commands

Sr No	Command	Meaning
1	Git init	This will initialise hidden and empty directory with name .git.
2	Git status	This gives status of files
3	Git add	This will add file from WD to staging area.
4	Git commit	This will add file from staging area to local repository
5	Git log	This gives information about number of commits
6	Git log --onefile	This gives information about commit ID
7	Git ls-files	This gives information about files in staging area
8	ls	This is linux command which gives information about list of files in WD

Version Control System Commands

Sr No	Command	Meaning
1	Git diff	This is used to compare two files on different components
2	Git rm	This is used to remove files from various component
3	Git checkout	This is used to discard unstaged changes in the tracked file of WD
4	Git reset	This is used to perform operation on WD, staging area and repository

RM Command

Sr No	Task	Command
1	Create file a.txt then add it to staging area and then commit it	Touch, add, commit
2	Create file b.txt then add it to staging area and then commit it	Touch, add, commit
3	Create file c.txt then add it to staging area and then commit it	Touch, add, commit
4	Remove file b.txt from WD and Staging Area	$ git rm b.txt
5	Remove file a.txt from Staging Area but not from WD	$ git rm --cached a.txt
6	Remove file c.txt from WD but not from Staging Area	$ rm c.txt

Checkout Command

Sr No	Task	Command
1	Create file a.txt and add some content to file	
2	add a.txt to staging area and commit it	
3	Modify content of a.txt (Add few lines)	
4	Discard unstaged(modified content)	$ git checkout -- a.txt

Reset Command

Sr No	Task	Command
1	Create file a.txt then add it to staging area and then commit it	Touch, add, commit
2	Create file b.txt then add it to staging area and then commit it	Touch, add, commit
3	Create file c.txt then add it to staging area and then commit it	Touch, add, commit
	Create file d.txt then add it to staging area and then commit it	Touch, add, commit
4	Discard file d.txt from local repository and staging area	$ git reset --mixed commmitID
5	Discard file c.txt only from local repository	$ git reset --soft commmitID
6	Discard file a.txt from everywhere	$ git reset --hard commmitID

Sr No	Task	Command
1	Create file a.txt then add it to staging area and then commit it	Touch, add, commit
2	Create file b.txt then add it to staging area and then commit it	Touch, add, commit
4	Create new branch with name branch1	$ git branch Branch1
5	switch to branch1 (Make branch1 as active)	$ git checkout Branch1
6	Create file x.txt then add it to staging area and then commit it	Touch, add, commit
7	switch to main branch	$ git checkout master
8	Create file c.txt then add it to staging area and then commit it	Touch, add, commit
9	check file status in between on all component	ls, git ls-files, git log --oneline

- It is used to combine changes from new branch to main branch

Fast Forward Merging:

There will not be any change in master branch from period of creation of new branch till the merging on new branch to master branch.

There is no possibility of conflict in Fast forward merging

Three Way Merge:

In this type changes will be there in master branch from period of creation of new branch till the merging on new branch to master branch.

There is possibility of conflict in Fast forward merging

Fast Forward Merging

Sr No	Task	Command
1	Create file a.txt then add it to staging area and then commit it	Touch, add, commit
2	Create file b.txt then add it to staging area and then commit it	Touch, add, commit
4	Create new branch with name F1 and switch to branch F1	branch, checkout
5	Create, add and commit 2 files x.txt, y.txt,	Touch, add, commit
6	Create file x.txt then add it to staging area and then commit it	Touch, add, commit
7	check logs on both branch	$ git checkout master
8	Switch to master branch and merge branch F1	Checkout, $ git merge F1
9	check logs on both branch	ls, git ls-files, git log --oneline

Three Way Merging

Sr No	Task	Command
1	Create file a.txt then add it to staging area and then commit it	Touch, add, commit
2	Create file b.txt then add it to staging area and then commit it	Touch, add, commit
4	Create new branch with name F1 and switch to branch F1	branch, checkout
5	Create, add and commit 2 files x.txt, y.txt	Touch, add, commit
7	check logs on both branch	ls, git ls-files, git log --oneline
8	Switch to master branch and create, add, commit file c.txt	checkout
9	Merge branch F1 and master and check logs in between	

Conflict Resolving

Sr No	Task	Command
1	Create file a.txt then add it to staging area and then commit it	Touch, add, commit
2	Modify contents of file a.txt then again add and commit	Touch, add, commit
4	Create new branch with name F1 and switch to branch F1	branch, checkout
5	Modify contents of file a.txt then again add and commit	Touch, add, commit
6	Switch to branch Master	checkout
7	Modify contents of file a.txt then again add and commit	Touch, add, commit
8	Merge Both branches together	Git merge F1
9	Resolve conflict and then merge again	

Rebase

- Rebase is another way to integrate changes from one branch to another.
- Rebase compresses all the changes into a single “patch.” Then it integrates the patch onto the target branch.
- Unlike merging, rebasing flattens the history because it transfers the completed work from one branch to another.

Syntax: Git rebase branchname

Sr No	Task	Command
1	Create file a.txt then add it to staging area and then commit it	Touch, add, commit
2	Create file b.txt then add it to staging area and then commit it	Touch, add, commit
3	Create new branch with name F1 and switch to branch F1	branch, checkout
4	Rebase F1 to master	Git rebase master
5	check logs	ls, git ls-files, git log --oneline
6	Switch to master	checkout
7	Merge Both branches together	Git merge F1
8	check logs and total files	ls, git ls-files, git log --oneline

- Github is hosting service for git repositories.
- Github is reaccommodate to store final copy of code

- Important Operations:

1. **Clone**: it is use to copy entire repository from remote machine to local machine.
   Syntax: Git clone “URL”

2. **Push**: It is use to upload files from local repository to remote repository
   Syntax: Git push origin main/master

3. **Pull**: It is use to download updates from remote repository to local repository
   Syntax: Git pull origin main/master

GitHub Command Line

Sr No	Task	Command
1	Login to git hub account, Create new repository	
2	Create file myFile add some content and commit it	
3	Initialize empty repository on local machine	
4	Download (clone) remote repository to local repository	Git clone “URL”
5	Modify content of myFile and Create newFile and add content	
6	add all files to staging area and then commit all files	
7	Push files to remote repository	Git push origin master
8	Change content of second file from remote repository	
9	Pull updates to local repository and check changes	Git pull origin master

Sr No	Task	Command
1	Open IntelliJ and create version control project	
2	Clone remote repository to IDE	Create project with VCS and clone
3	Modify content of myFile and Create newFile and add content	
4	add all files to staging area and then commit all files	RightClick ->git -> add ->commit
5	Push files to remote repository	RightClick ->git -> push
6	Change content of second file from remote repository	
7	Pull updates to local repository and check changes	RightClick ->git -> pull

OPERATIVE

January 2023