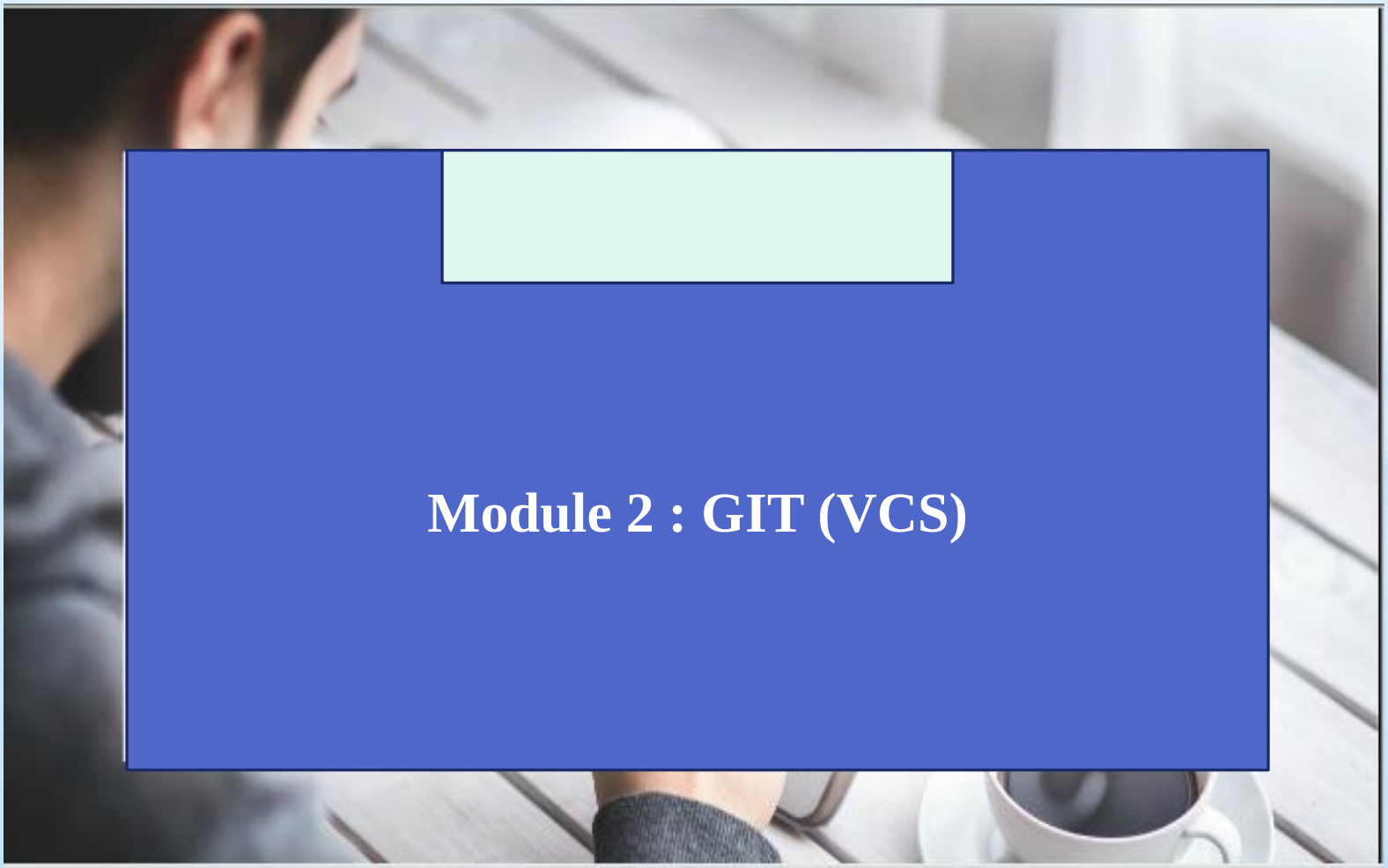
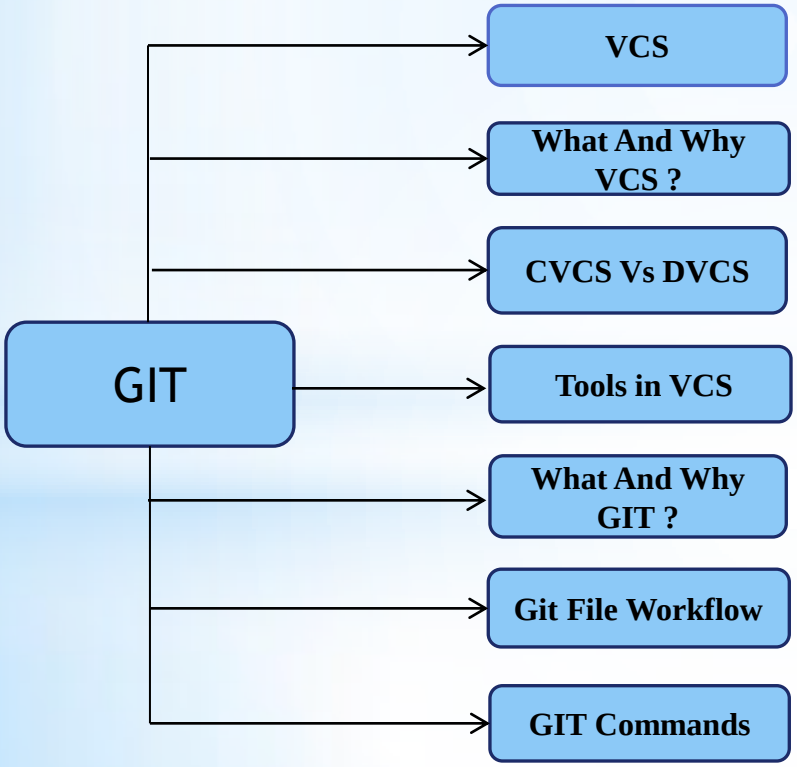


Module 2 : GIT (VCS)


Git (VCS)

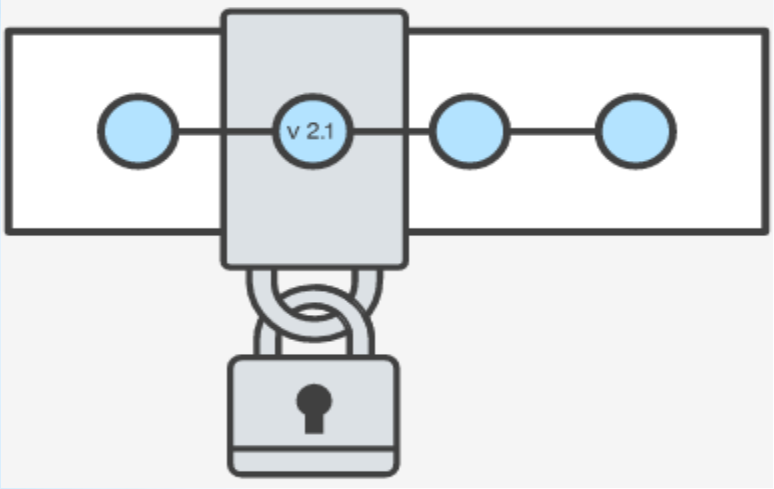
Objective Of Session



Version Control System(GIT) Definitions

1. What is Version Control System

- ✓ Version control systems are a category of software tools that help a software team manage changes to source code over time
- ✓ Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made,
- ✓ Developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members



Benefits of Version Control System

Centralized Version Control System

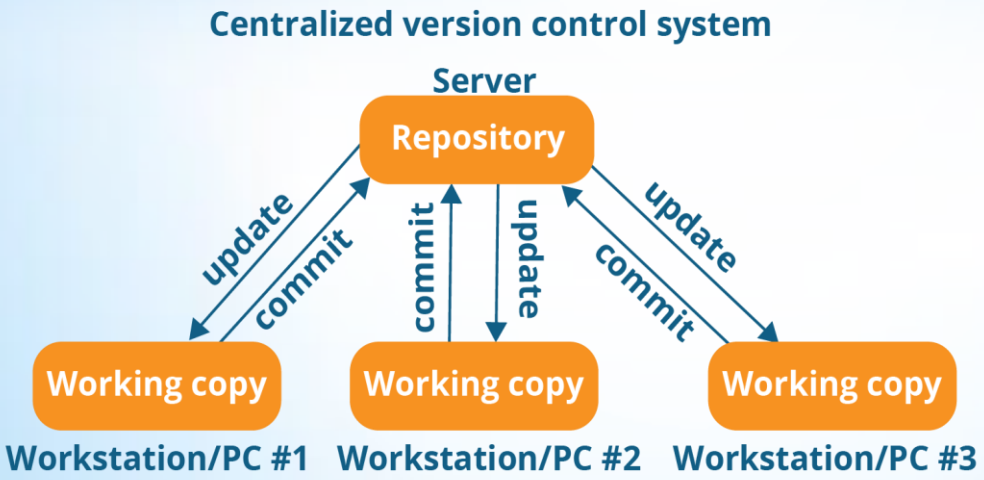
Types of VCS

1. Centralized Version Control System
2. Distributed Version Control System

CVCS Vs. DVCS

Centralized version control system (CVCS) :

Uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server.
Please refer to the diagram below to get a better idea of CVCS:



Centralized Version Control System

The repository in the above diagram indicates a central server that could be local or remote which is directly connected to each of the programmer's workstation.
Every programmer can extract or update their workstations with the data present in the repository or can make changes to the data or commit in the repository. Every operation is performed directly on the repository.
Even though it seems pretty convenient to maintain a single repository, it has some major drawbacks.

Drawbacks in CVCS

- ✓ It is not locally available; meaning you always need to be connected to a network to perform any action
- ✓ Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.

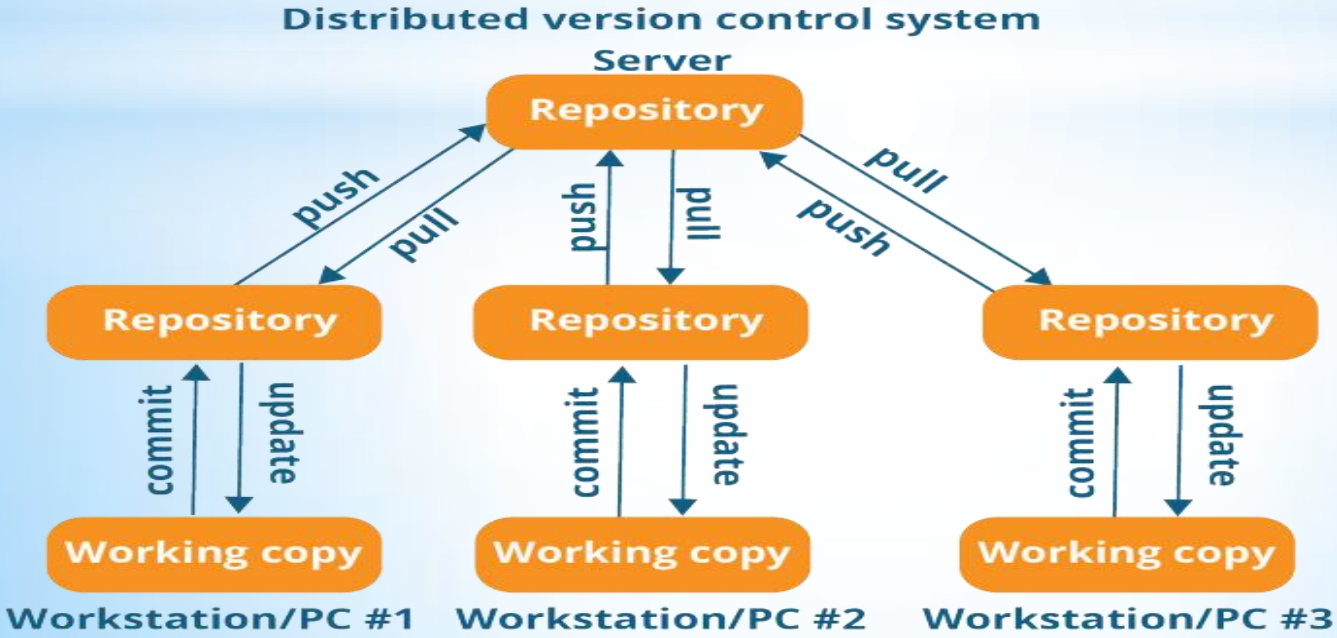
This is when Distributed VCS comes to the rescue.

Distributed Version Control System

Distributed Version Control System

These systems do not necessarily rely on a central server to store all the versions of a project file. In Distributed VCS, every contributor has a local copy or “clone” of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.

You will understand it better by referring to the diagram below:



Distributed Version Control System

As you can see in the above diagram, every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive. They can commit and update their local repository without any interference.
They can update their local repositories with new data from the central server by an operation called "**pull**" and affect changes to the main repository by an operation called "**push**" from their local repository.
The act of cloning an entire repository into your workstation to get a local repository gives you the following advantages:

Advantages of DVCS

- ✓ All operations **(except push & pull**) are very fast because the tool only needs to access the hard drive, not a remote server. Hence, you do not always need an internet connection.

- ✓ Committing new change-sets can be done locally without manipulating the data on the main repository. Once you have a group of change-sets ready, you can push them all at once.

- ✓ Since every contributor has a full copy of the project repository, they can share changes with one another if they want to get some feedback before affecting changes in the main repository.

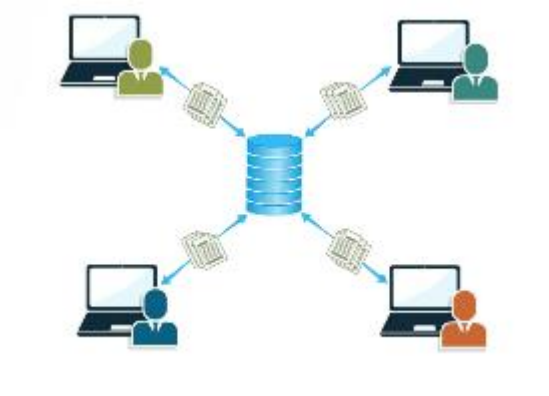
- ✓ If the central server gets crashed at any point of time, the lost data can be easily recovered from any one of the contributor's local repositories.

After knowing Distributed VCS, its time we take a dive into what is Git.

Benefits of Version Control System

Archived version of the files are not available

A Version control system or VCS, is a system that allows multiple users to manage multiple revisions of the same unit of information. It is the snapshot of your project over time



GIT Definitions

What Is Git?

Git is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software. Before you go ahead, check out this video on GIT which will give you better in-sight

Git provides with all the Distributed VCS facilities to the user that was mentioned earlier. Git repositories are very easy to find and access. You will know how flexible and compatible Git is with your system

GIT Features

Free and Open Source

Speed

Scalable

Reliable

Distribute Development

Secure

Economical

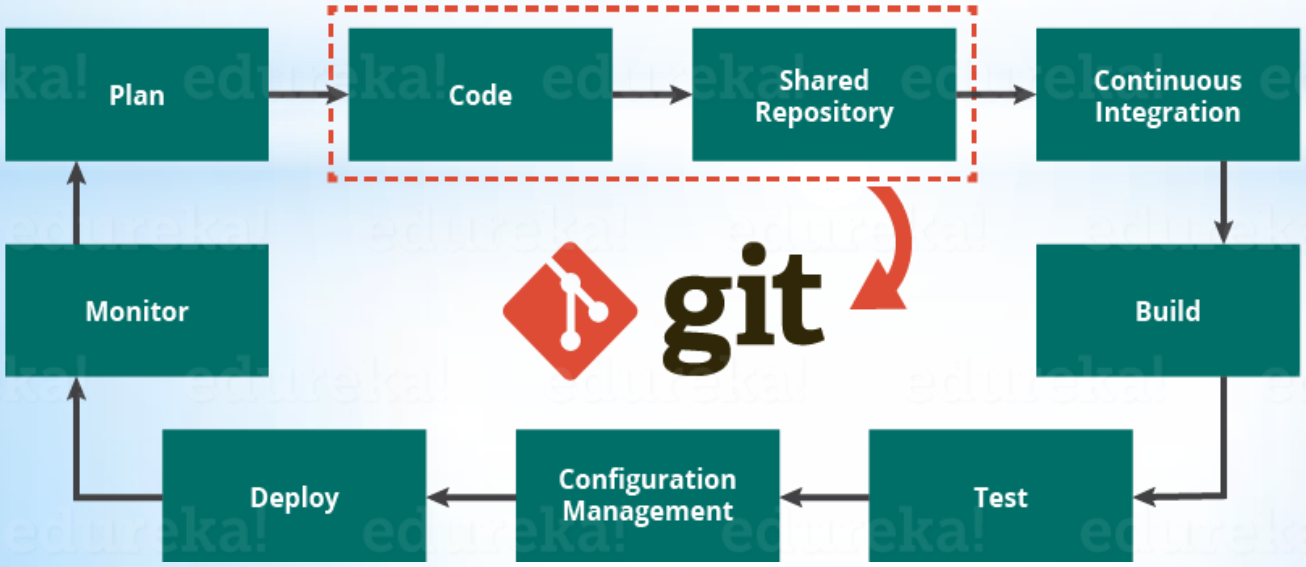
Supports non-linear Development

Easy Branding

Compatibility with existing system or protocol

Role of GIT In DevOps?

Now that you know what is Git, you should know Git is an integral part of DevOps.
DevOps is the practice of bringing agility to the process of development and operations. It's an entirely new ideology that has swept IT organizations worldwide, boosting project life-cycles and in turn increasing profits. DevOps promotes communication between development engineers and operations, participating together in the entire service life-cycle, from design through the development process to production support.
The diagram below depicts the Devops life cycle and displays how Git fits in Devops.



Git Repositories

A Git repository contains the history of a collection of files starting from a certain directory. The process of copying an existing Git repository via the Git tooling is called _cloning. After cloning a repository the user has the complete repository with its history on his local machine. Of course, Git also supports the creation of new repositories.

Imp Points in GIT repository

- ✓ If you want to delete a Git repository, you can simply delete the folder which contains the repository.
- ✓ If you clone a Git repository, by default, Git assumes that you want to work in this repository as a user.

- ❑ **Git also supports the creation of repositories targeting the usage on a server.**

- ❑ **Bare Repositories :**

bare repositories are supposed to be used on a server for sharing changes coming from different developers. Such repositories do not allow the user to modify locally files and to create new versions for the repository based on these modifications.

- ❑ **Non-bare repositories :**

non-bare repositories target the user. They allow you to create new changes through modification of files and to create new versions in the repository. This is the default type which is created if you do not specify any parameter during the clone operation.

Git Working Tree

A local repository provides at least one collection of files which originate from a certain version of the repository. This collection of files is called the *working tree*. It corresponds to a checkout of one version of the repository with potential changes done by the user.
The user can change the files in the *working tree* by modifying existing files and by creating and removing files. A file in the working tree of a Git repository can have different states.

These states are the following:

- untracked: the file is not tracked by the Git repository. This means that the file never staged nor committed.
- tracked: committed and not staged
- staged: staged to be included in the next commit
- dirty / modified: the file has changed but the change is not staged

Key Point

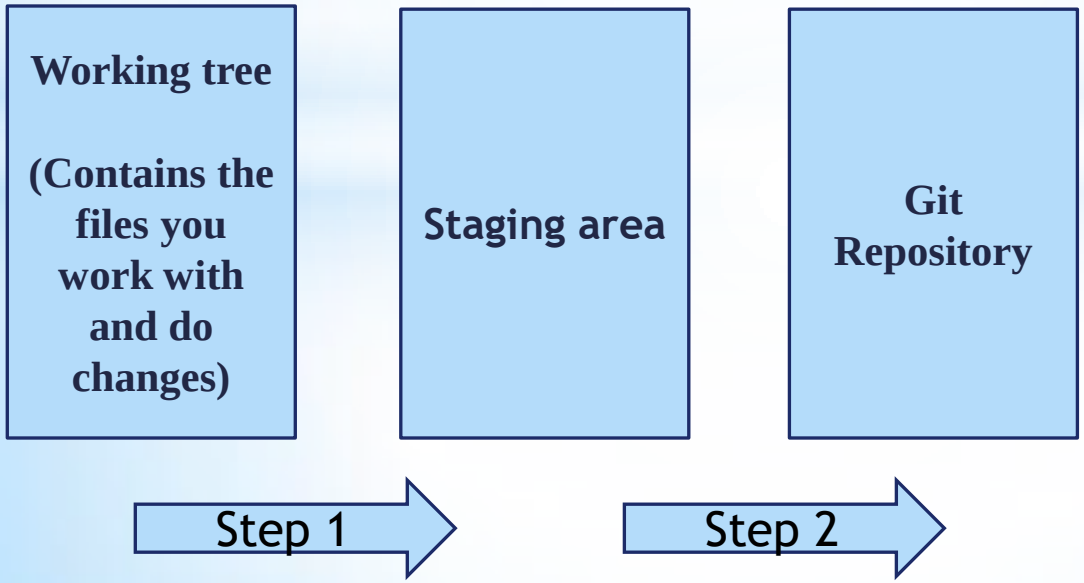
After doing changes in the working tree, the user can add these changes to the Git repository or revert these changes.

Adding a git repository via staging and Committing

After modifying your *working tree* you need to perform the following two steps to persist these changes in your local repository

- add the selected changes to the *staging area* (also known as index) via the **git add** command
- commit the staged changes into the Git repository via the **git commit** command

This process is depicted in the following graphic.



The Concept of Branches

Git supports *branching* which means that you can work on different versions of your collection of files. A branch allows the user to switch between these versions so that he can work on different changes independently from each other.

- For example, if you want to develop a new feature, you can create a branch and make the changes in this branch. This does not affect the state of your files in other branches. For example, you can work independently on a branch called *production* for bug fixes and on another branch called *feature_123* for implementing a new feature.

- Branches in Git are local to the repository. A branch created in a local repository does not need to have a counterpart in a remote repository. Local branches can be compared with other local branches and with _remote-tracking branches. A remote-tracking branch proxies the state of a branch in another remote repository.

- Git supports the combination of changes from different branches. The developer can use Git commands to combine the changes at a later point in time

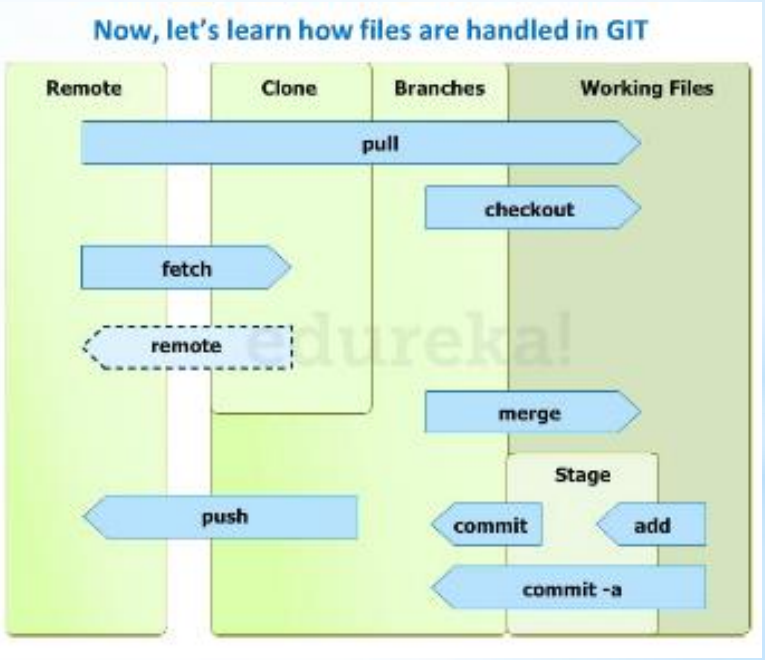
Why GIT?

The above graph shows the usability of the popular VCS tools in the past 5 years

Reference: https://www.google.co.in/trends/explore?date=today%2012-m&q=Git,SVN,perforce,mercurial

GIT File Workflow

GIT is a free and open and source distributed VCS
Used in software development with emphasis On speed, data integrity, and support for Distributed, non-linear workflow



Why GIT is Winner in Market ?

1. Distributed Nature

- ✓ Every user has a complete copy of the repository data stored locally
- ✓ It allows full functionality when disconnected from the network
- ✓ Due to being distributed, you don't have to give the commit access to team members to use & access the version future

2. Branch Handling

- ✓ In GIT every developer's working directory is itself a branch
- ✓ Git tracks project revision of the branch
- ✓ Records branch & Merge events

Why GIT is Winner in Market ?

Better Merging

- ✓ Two GIT users can merge changes with each other and the push the changes to the remote repository
- ✓ GIT automatically starts the next merge at the last merge
- ✓ In GIT, you decide when to merge what from whom

GIT Plugins

- ✓ GitFlow- This plugin permits to manage branching model in GIT
- ✓ EGit- This plugin implements Eclipse tool on top of Java implementation of GIT
- ✓ Git-Client- This plugin allows use of GIT as a build SCM. Interaction with the GIT run time is performed by the use of the git-client plugin

Why GIT is Winner in Market ?

Better Merging

- ✓ Two GIT users can merge changes with each other and the push the changes to the remote repository
- ✓ GIT automatically starts the next merge at the last merge
- ✓ In GIT, you decide when to merge what from whom

GIT Plugins

- ✓ GitFlow- This plugin permits to manage branching model in GIT
- ✓ EGit- This plugin implements Eclipse tool on top of Java implementation of GIT
- ✓ Git-Client- This plugin allows use of GIT as a build SCM. Interaction with the GIT run time is performed by the use of the git-client plugin

GIT Hub

Topics of GitHub :

- What is GitHub?
- Create a Repository
- Create a Branch
- Make a Commit

What is GitHub?

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

What is Repository

Repository is place where store your all working files into that

Installation of GIT and GitHub

Installation of Git & GitHub

Let me guide you through the process to install Git in your system following by **Golden Technology** document use case

- How to install Git in Windows
- How to install Git in Centos
- How to create a GitHub Account
- How to make your repositories on GitHub

We need to follow the steps in specified documents

Important GIT Commands

Some of the important GIT commands that one must know when working in a **Local Repository**

1. Initialization – Create a new local repository **git init**

2. Add files

- Add one file for staging **add <file name>**
- Add multiple files for staging **git add .**

3. Status - To check the status of the repository **git status**

4. Commit – To commit changes in the HEAD of the local repository **git commit –m "commit message"**

– To commit changes without staging the file **git commit –a –m "commit message"**

5. History – To see the commit history **git log**

Important GIT Commands

6. Branch

- Create new branch **git branch <branchname>**
- To switch from one to another **git checkout <branchname>**
- To switch from any to master **git checkout master**

7. Merge – To merge changes to repository **git merge <sourcebranch> <Destination branch>**

8. Stash

- Save the local modifications(without commit) and revert to the working directory to match the HEAD commit **git stash**
- Switch to the branch where the changes were saved **git stash apply**

Important GIT Commands

Some of the important GIT commands that one must know when working in a **Remote Repository**

1. **Existing Repo -** To list the existing remote repositories **git remote**
2. **Clone** – Create a working copy in a local repository **git clone /path/to/repository**
3. **Fetch** – To fetch the remote repository changes since you last cloned/fetched **git fetch origin**
4. **Pull** – To fetch and merge changes from remote repository to working directory **git pull origin**
5. **Push** – To push changes to the master branch of your remote repository **git push origin master**

Git END