



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

Assistant Professor  
BITS Pilani

# Agenda

## Could for DevOps & Version Control System

- Cloud as a catalyst for DevOps
- Evolution of Version Control
- Version Control System Types
  - Centralized Version Control Systems
  - Distributed Version Control Systems
- Introduction to GIT
- GIT Basics commands
- Creating Repositories, Clone, Push, Commit, Review
- Git Branching
- Git Managing Conflicts
- Git Tagging
- Git workflow
  - Centralized Workflow
  - Feature Branch Workflow
- Best Practices- clean code



# DevOps Tools

## Cloud as a Catalyst for DevOps

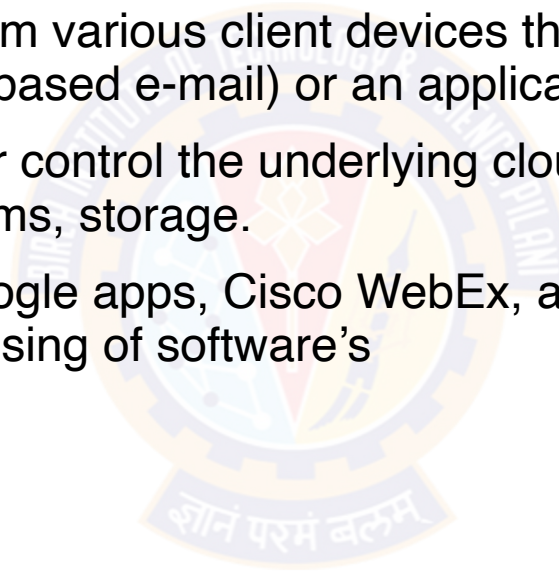
- Characterization of the cloud by National Institute of Standards and Technology (NIST)
  - On-demand self-service
  - Broad network access
  - Resource pooling
  - Rapid elasticity
    - It is the Capabilities can be elastically provisioned and released
- Measured service
- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)



# Cloud as a Catalyst for DevOps

## Software as a Service (SaaS)

- In this the consumer is provided the capability to use the provider's applications running on a cloud infrastructure
- The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based e-mail) or an application interface
- The consumer does not manage or control the underlying cloud infrastructure including networks, servers, operating systems, storage.
- For an example, you can relate google apps, Cisco WebEx, as a Service, Office 365 service, where Provider deals with the licensing of software's

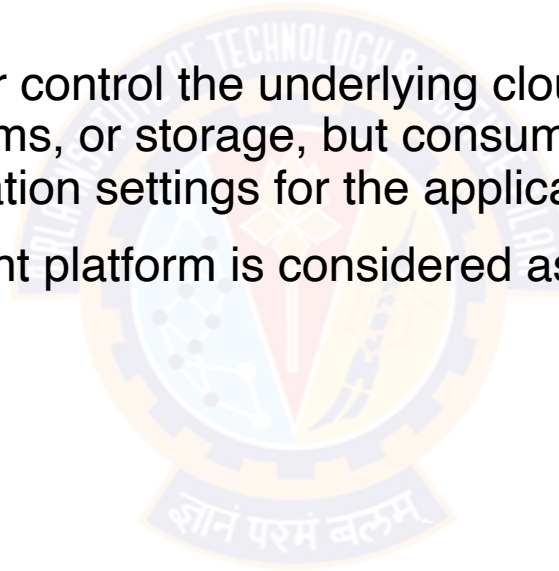




# Cloud as a Catalyst for DevOps

## Platform as a Service (PaaS)

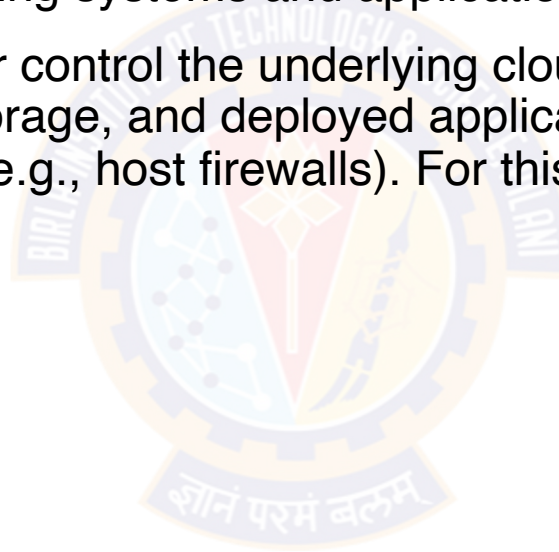
- The consumer is provided the capability to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider
- The consumer does not manage or control the underlying cloud infrastructure including networks, servers, operating systems, or storage, but consumer has control over the deployed applications and possibly configuration settings for the application-hosting environment
- For an Example: .NET Development platform is considered as a platform



# Cloud as a Catalyst for DevOps

## Infrastructure as a Service (IaaS)

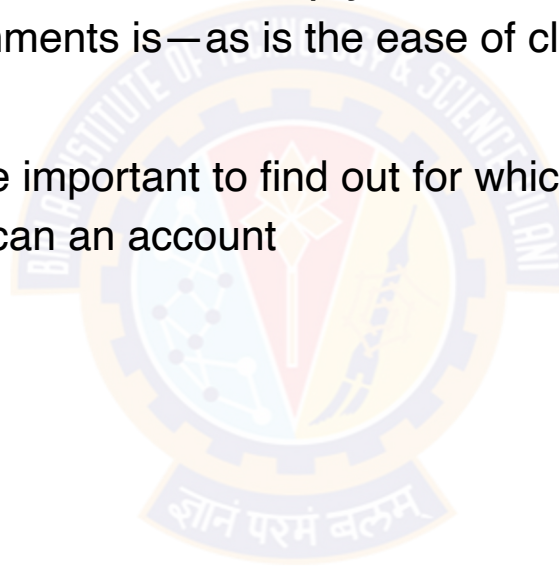
- The consumer is provided the capability to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications
- The consumer does not manage or control the underlying cloud infrastructure but consumer has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls). For this you can consider any Server Provisioning is IaaS,



# Cloud as a Catalyst for DevOps

## Three of the unique aspects of the cloud that impact DevOps

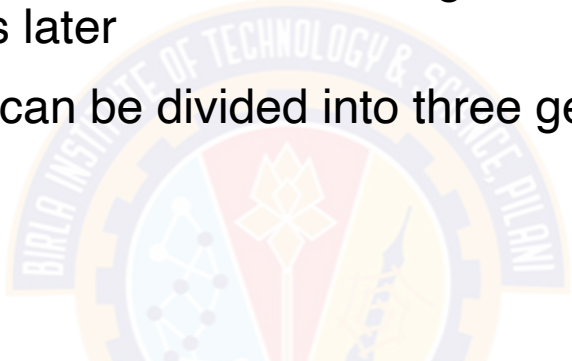
- Three of the unique aspects of the cloud that impact DevOps:
- The ability to create and switch environments simply
  - Simply create and migrate environments is—as is the ease of cloning new instances
- The ability to create VMs easily
  - Administering the running VMs are important to find out for which VM we are paying but not using it
  - Tool such as, Janitor Monkey to scan an account
- The management of databases



# Evolution of Version Control

## Generations of VCS

- What is “version control”, and why should you care?
- Definition: Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later
- The history of version control tools can be divided into three generations



Generation	Networking	Operations	Concurrency	Example Tool
First Generation	None	One file at a time	Locks	RCS, SCCS
Second Generation	Centralized	Multi-file	Merge before commit	CVS, Subversion
Third Generation	Distributed	Changesets	Commit before merge	Bazaar, Git



# Version Control System

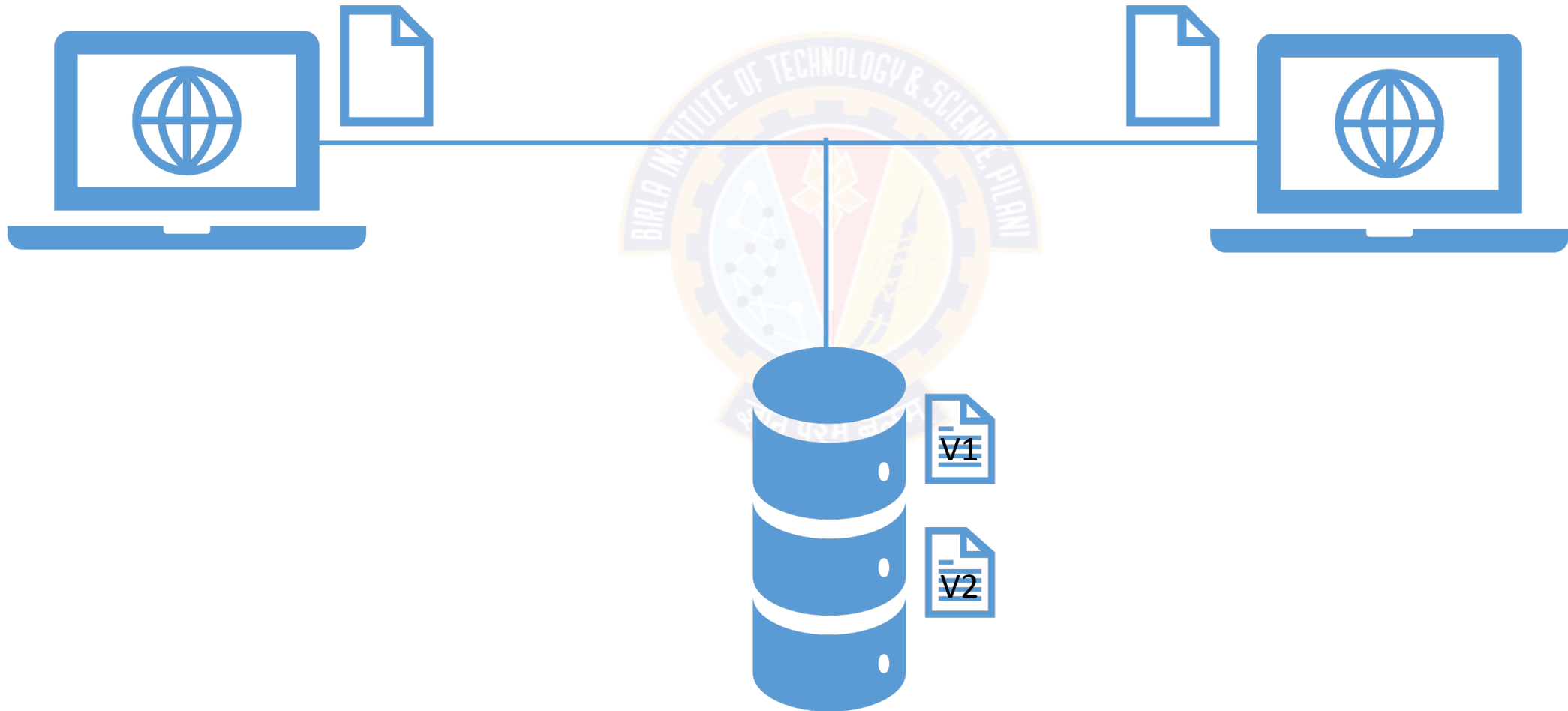
## Benefits

- Change history
- Concurrent working (Collaboration)
- Traceability
- Backup & Restoration



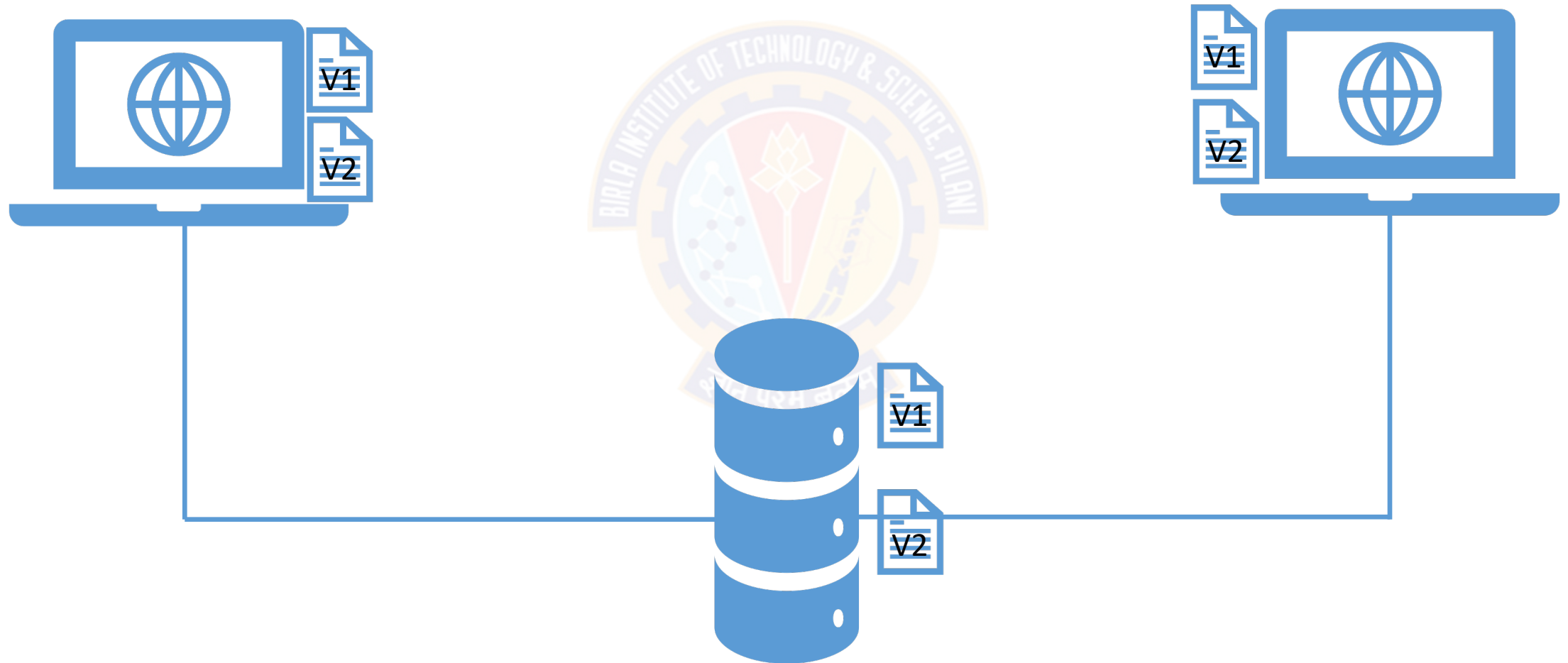
# Version Control System Types

## Centralized source code management System



# Version Control System Types

## Distributed source code management System



# CVCS Vs. DVCS

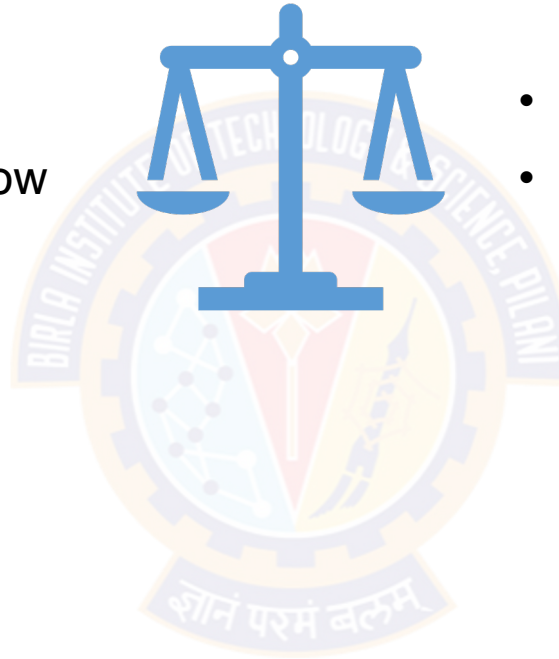
Lets discuss con's

## CVCS

- Single point of failure
- Remote commits are slow
- Continuous connection

## DVCS

- Need more space
- Bandwidth for large project



# Available Tools

## CVCS

- Open source:
  - Subversion (SVN)
  - Concurrent Versions System (CVS)
  - Vesta
  - OpenCVS
- Commercial:
  - AccuRev
  - Helix Core
  - IBM Rational ClearCase
  - Team Foundation Server (TFS)

## DVCS

- Open source:
  - Git
  - Bazaar
  - Mercurial
- Commercial:
  - Visual Studio: Team Services
  - Sun WorkShop: TeamWare
  - Plastic SCM – by Codice Software, Inc
  - Code Co-op





# Git & GitHub

## What we will learn in this session

- Git & GitHub relationship
- Prerequisite
- Foundation of Git



Concept of Git



Understanding GitHub



Beyond the basics

# Git

## What is Git



Popular Source Control system



Distributed system



Free (Open Source tool)



# Git

## Why use Git

**Fast**

**Disconnected**

**Powerful yet easy**

**Branching**

**Pull Requests**



# GitHub

## What is GitHub?



Hosting service on Git



More than just source control for your code

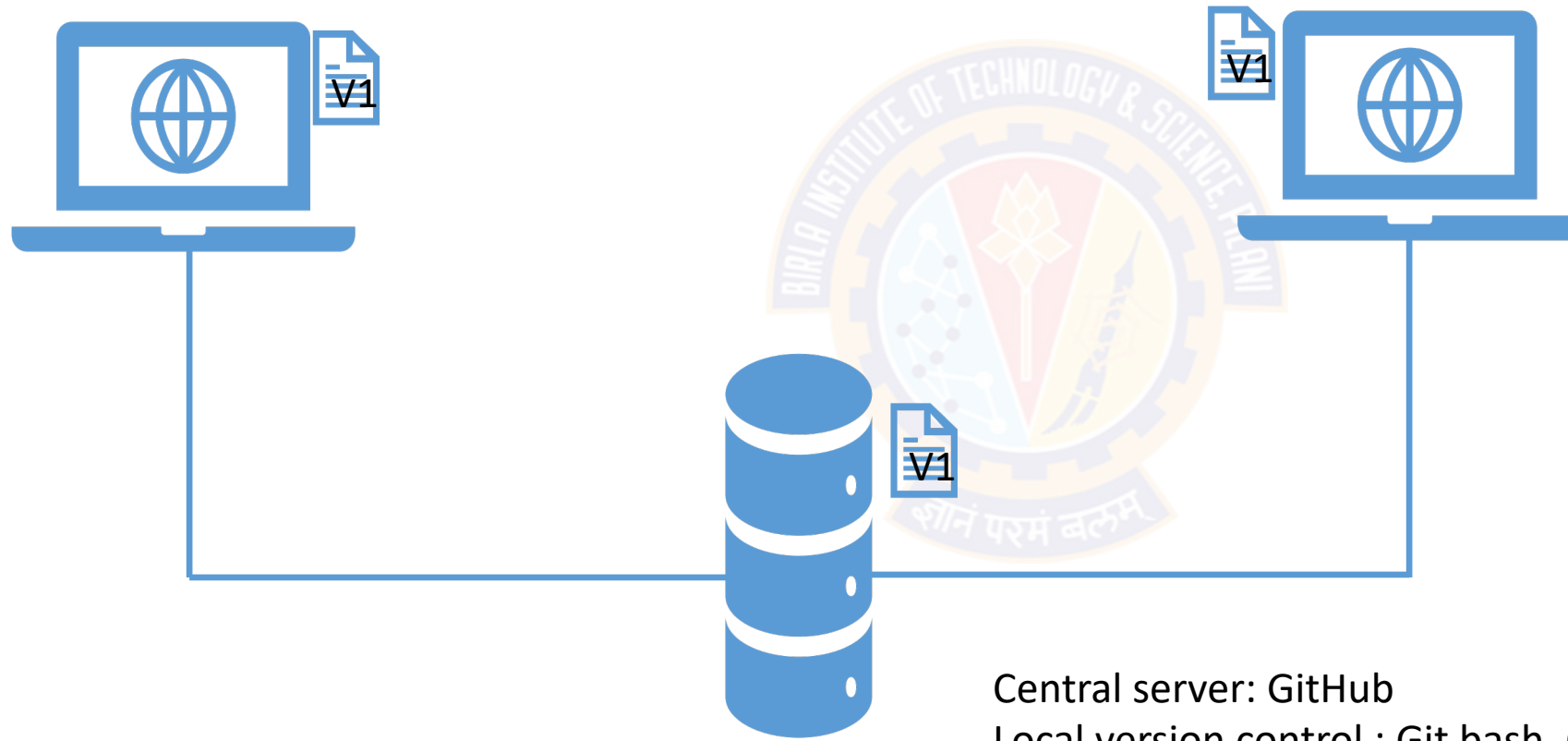
- Issue management, Working with Teams, etc.,



Free & Paid options

# Git & GitHub

## Relationship





# Git

## Working with Git



### Console

We will use this



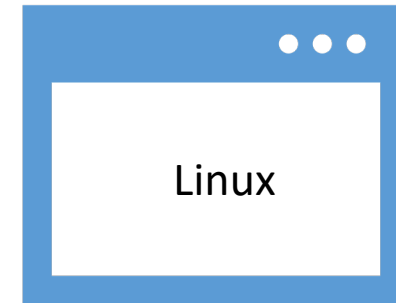
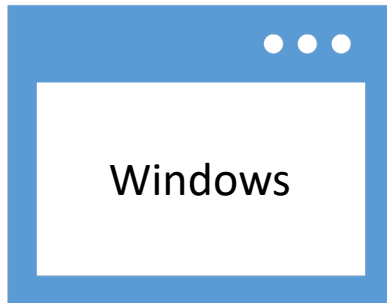
### GUI

GitHub Desktop  
Source tree

# Git & GitHub

## Getting your system ready

- Install appropriate Git bash form official site
- Command line (Git bash)
- Server account (GitHub account)



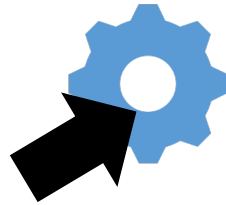
Support for all platform

# Git Foundation

## The 3 State of Git



Modified



Staged

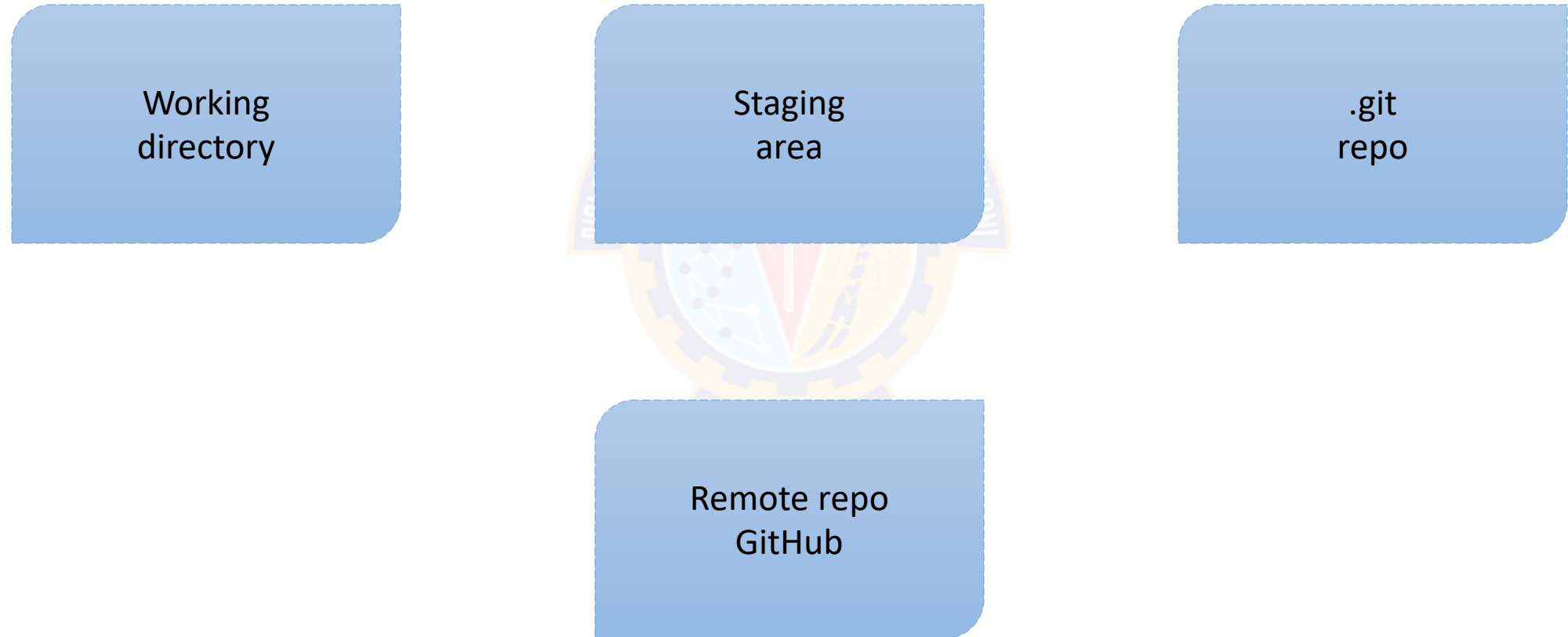


Committed



# Git Foundation

## The 3 areas of Git



# Git Foundation

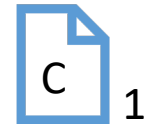
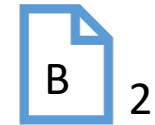
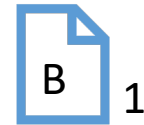
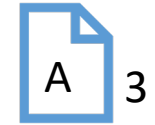
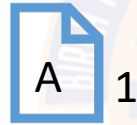
## Concepts of Snapshots in Git & GitHub

First  
version

Version 2

Version 3

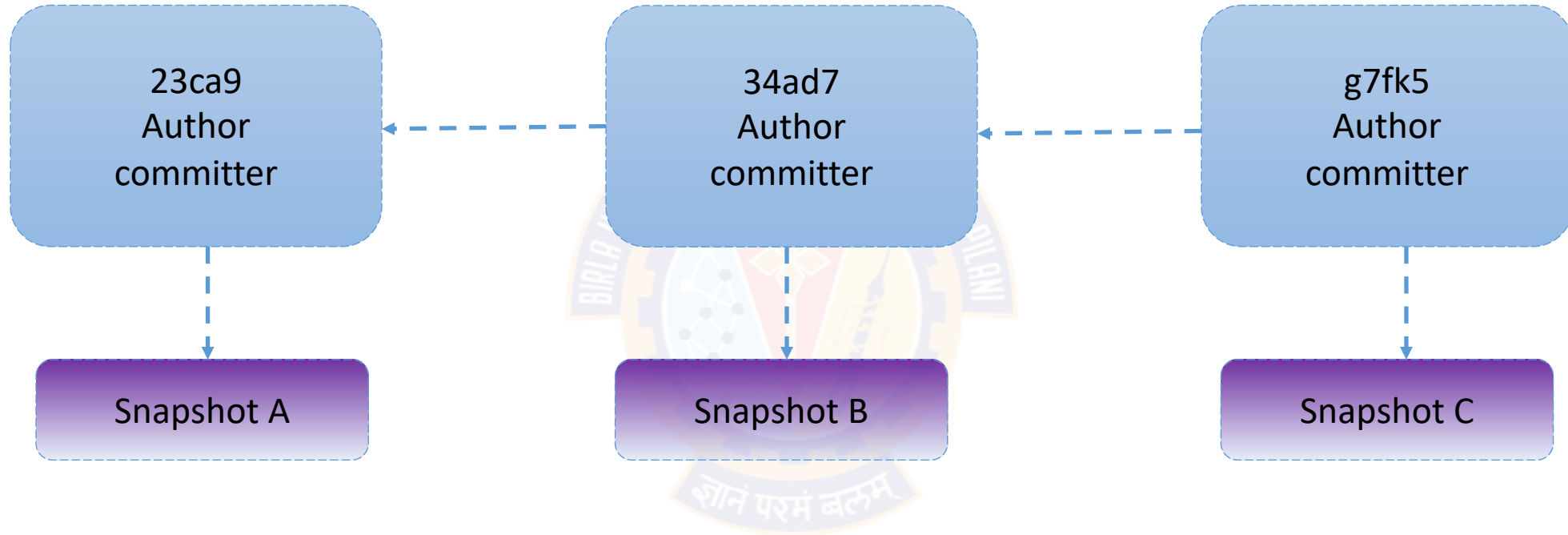
Version 4





# Git Foundation

## Commits in Git



# Git

## Basic Commands

\$ git	\$ git push
\$ git config	\$ git fetch
\$ git init	\$ git merge
\$ git clone	\$ git pull
\$ git status	\$ git log
\$ git add	\$ git reset
\$ git commit	\$ git revert
\$ git branch	
\$ git checkout	

# GitHub

## GitHub's main Features



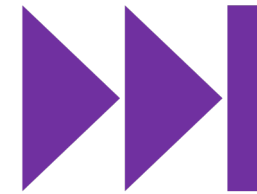
Code management



Pull requests



Issues



CI/CD



Global search

# GitHub

## Connecting with local machine

### HTTPS

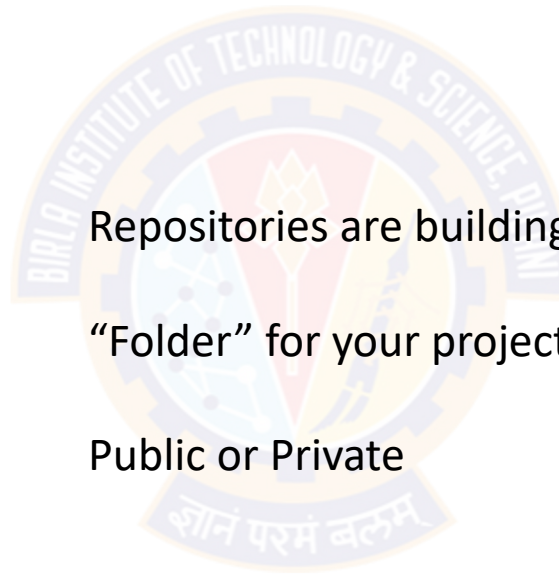
Requires user name & password

### SSH

Easier to work with



## Working with repository



Repositories are building block of GitHub

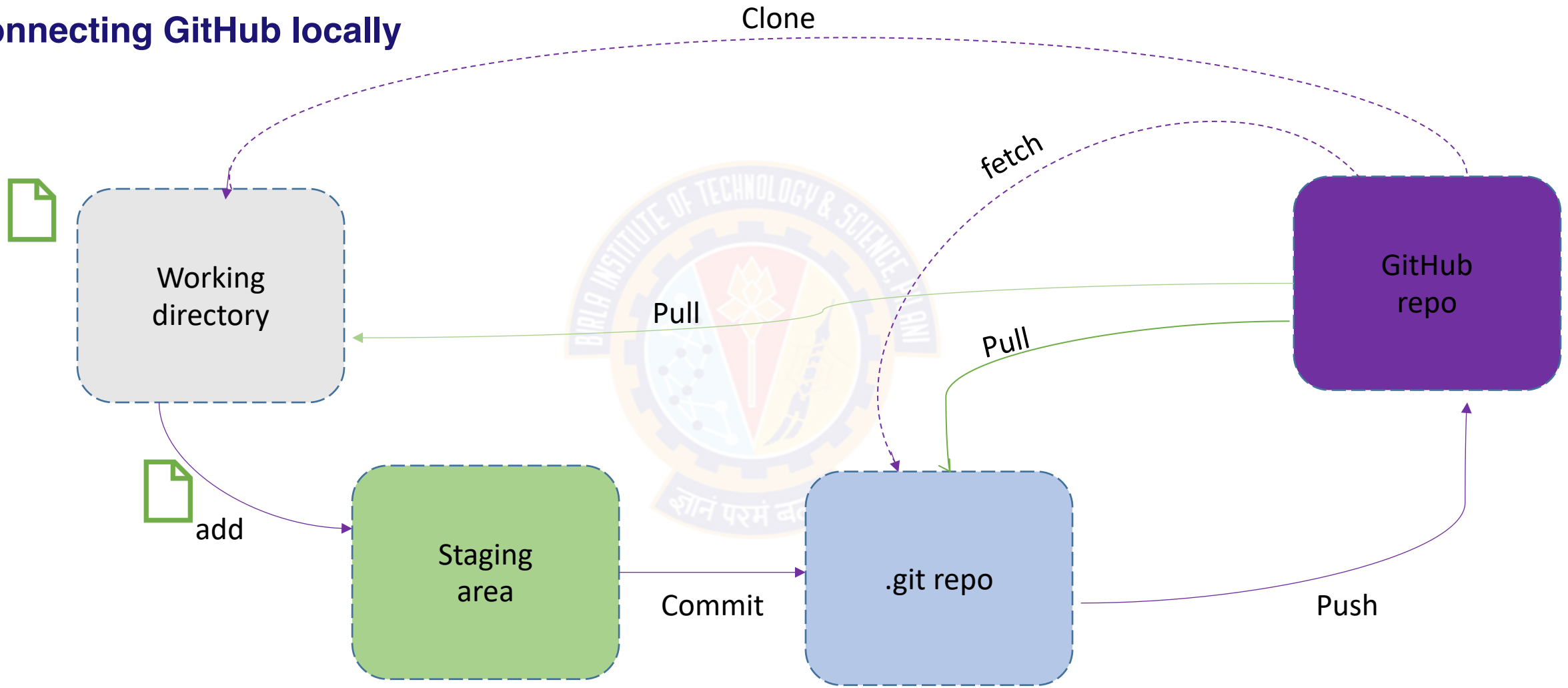
“Folder” for your project

Public or Private



# GitHub

## Connecting GitHub locally



# GitHub

## Working with special files



README is special file known by GitHub

Rendered automatically on landing page

Typically written in markdown(.md)

Other files:

LICENSE

CHANGELOG

CODE\_OF\_CONDUCT

CONTRIBUTORS

SUPPORT

CODEOWNERS

## Repository Feature

TOPICS

ISSUES

Insight

PROJECTS

Pull Requests

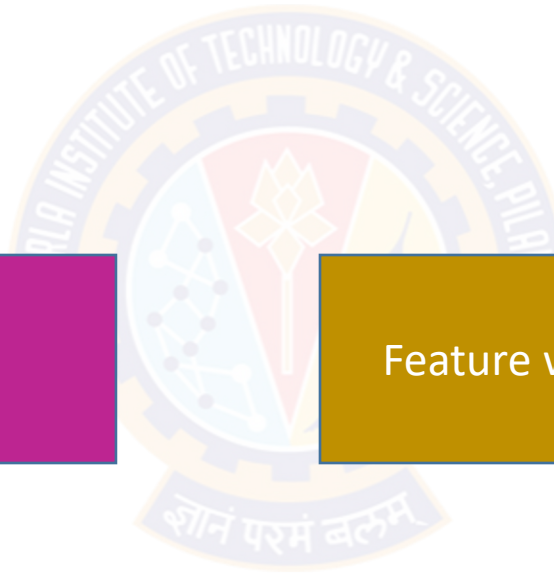
Settings

# Git

## Workflow

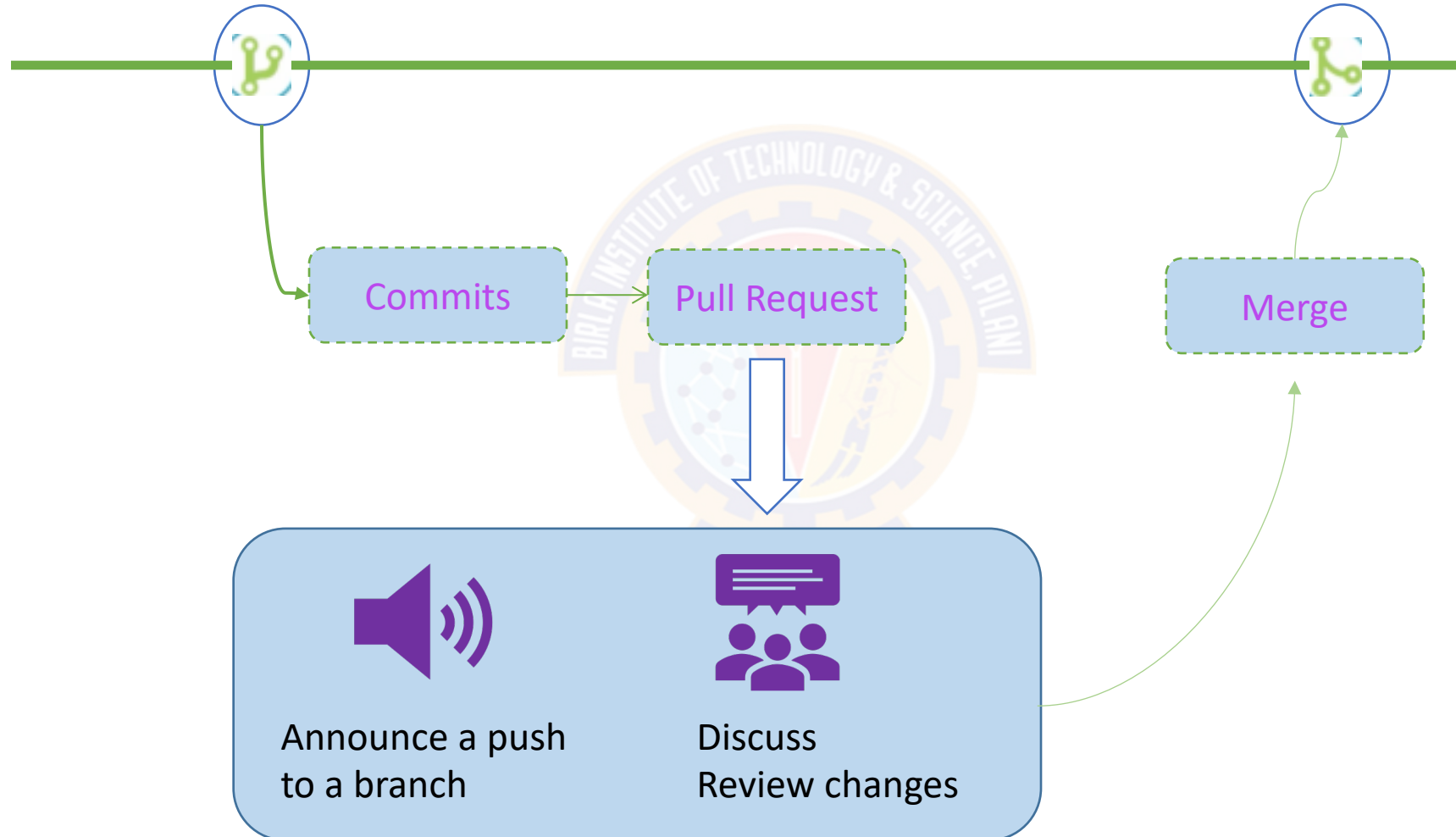
Centralized

Feature workflow



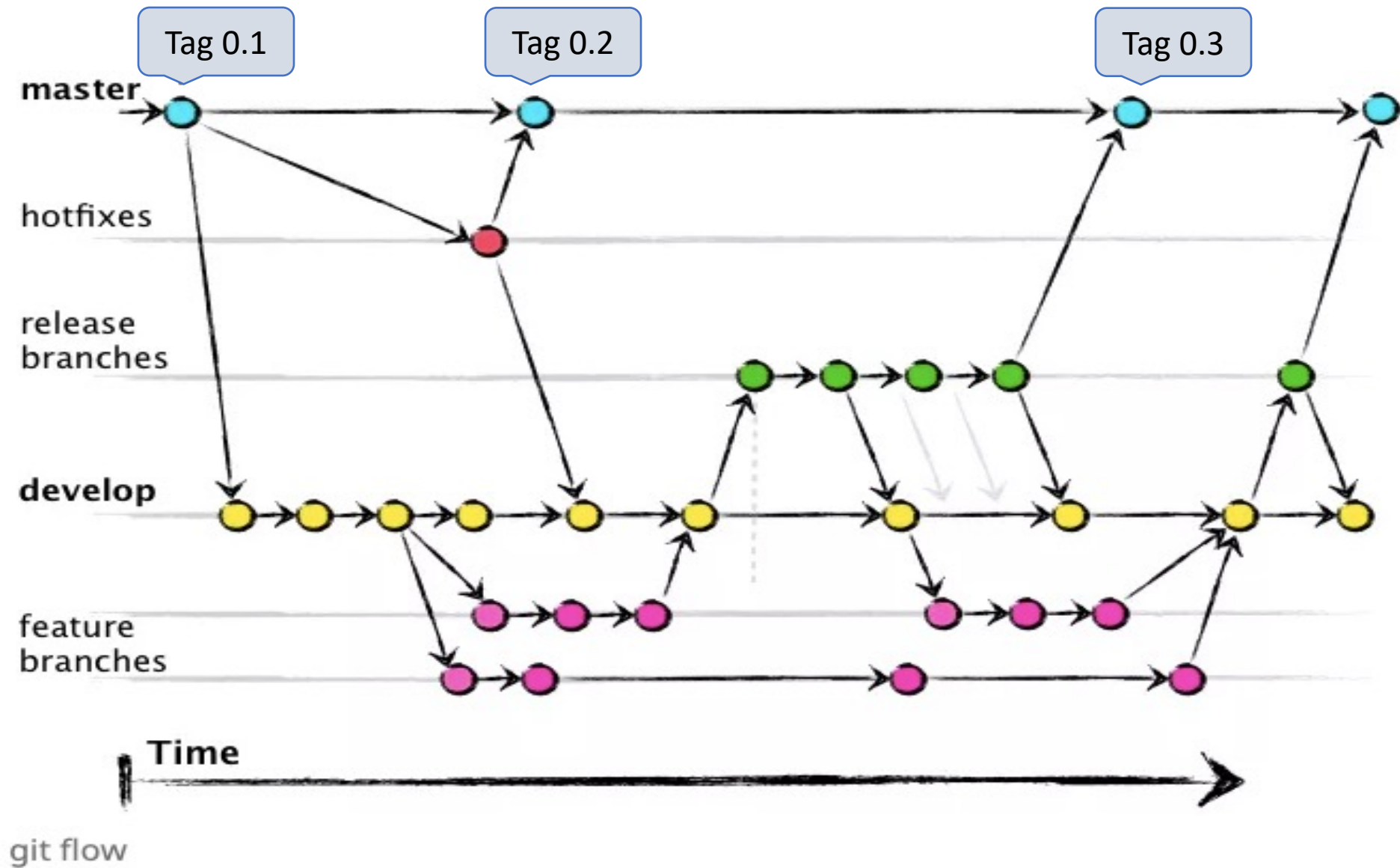
# GitHub

## Branching



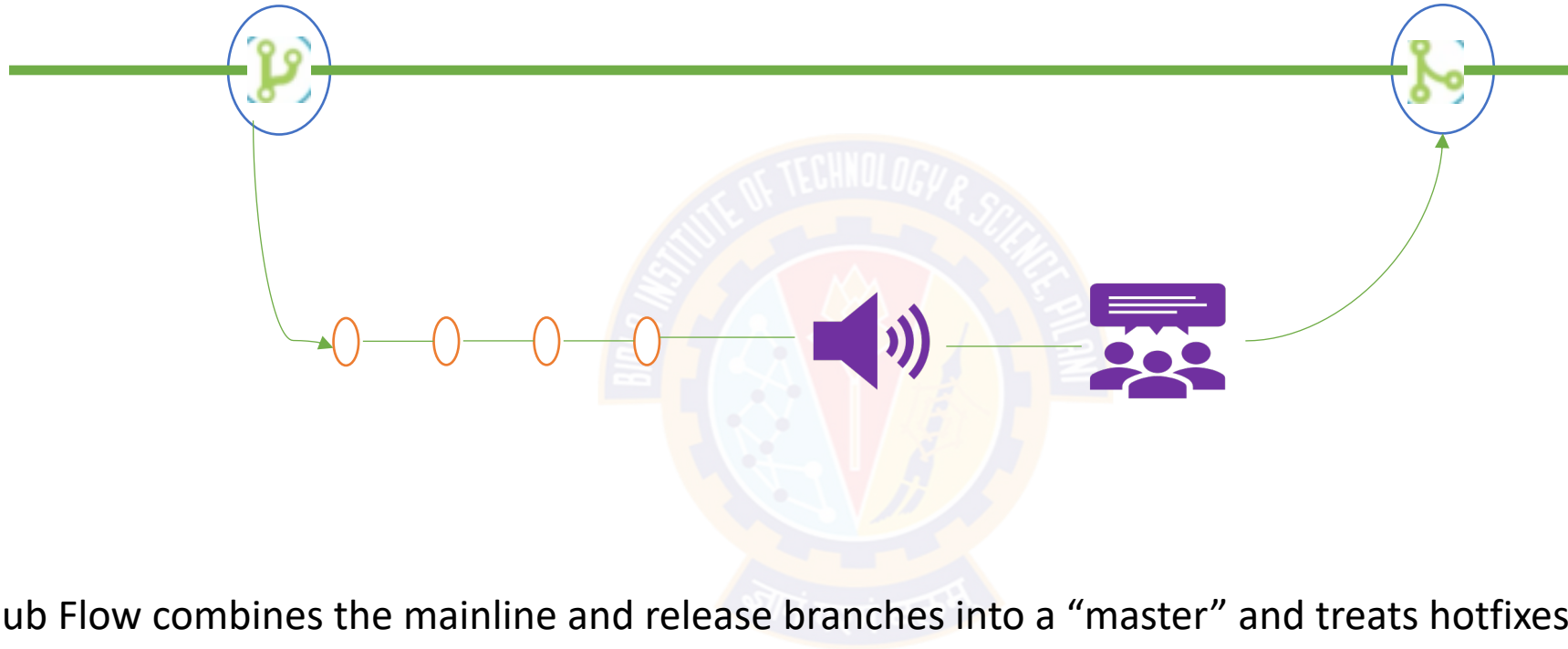
# GitHub

## Git Flow



# GitHub

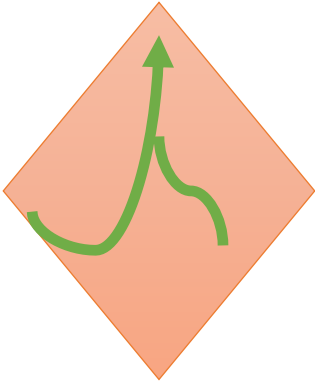
## GitHub Flow



GitHub Flow combines the mainline and release branches into a “master” and treats hotfixes just like feature branches.

# Git & GitHub

## Merging with conflicts



Typical conflicts:

- Editing on the same line

- Editing on already deleted file

Merge conflicts needs to be solved before merge happen (Manual intervention)



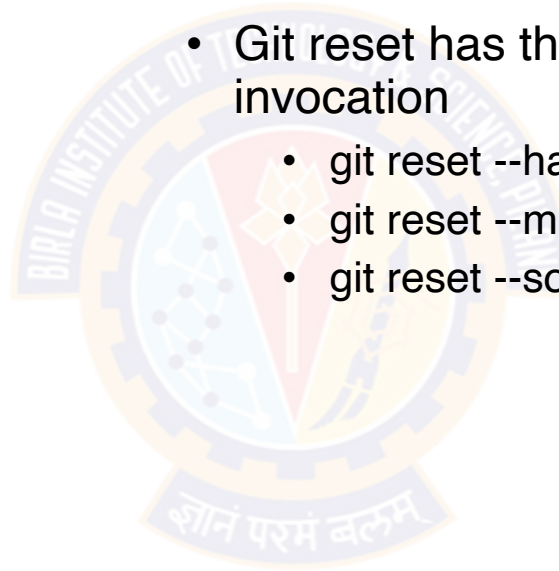
# Git Command

## Revert

- Git revert will create a new commit
- Git revert undoes a single commit
- It is a safe way if undo

## Reset

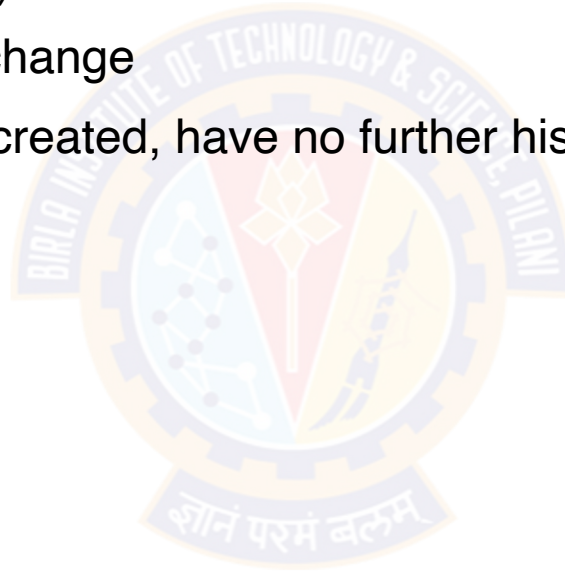
- Git reset command is a complex
- Dangerous
- Git reset has three primary form of invocation
  - `git reset --hard HEAD`
  - `git reset --mixed HEAD`
  - `git reset --soft HEAD`



# Git command

## Git Tag

- Tags are ref's that point to specific points in Git history
- Marked version release (i.e. v1.1.1)
- A tag is like a branch that doesn't change
- Unlike branches, tags, after being created, have no further history of commits
- Common Tag operations:
  - Create tag
  - List tags
  - Delete tag
  - Sharing tag



## Clean Code

- Follow standard conventions
- Keep it simple, Simpler is always better, Reduce complexity as much as possible
- Be consistent i.e. If you do something a certain way, do all similar things in the same way
- Use self explanatory variables
- Choose descriptive and unambiguous names
- Keep functions small
- Each Function should do one thing
- Use function names descriptive
- Prefer to have less arguments
- Always try to explain yourself in code; put proper comments
- Declare variables close to their usage
- Keep lines short
- Code should be readable
- Code should be fast
- Code should be generic
- Code should be reusable



# Q&A



# Thank You!

In our next session: