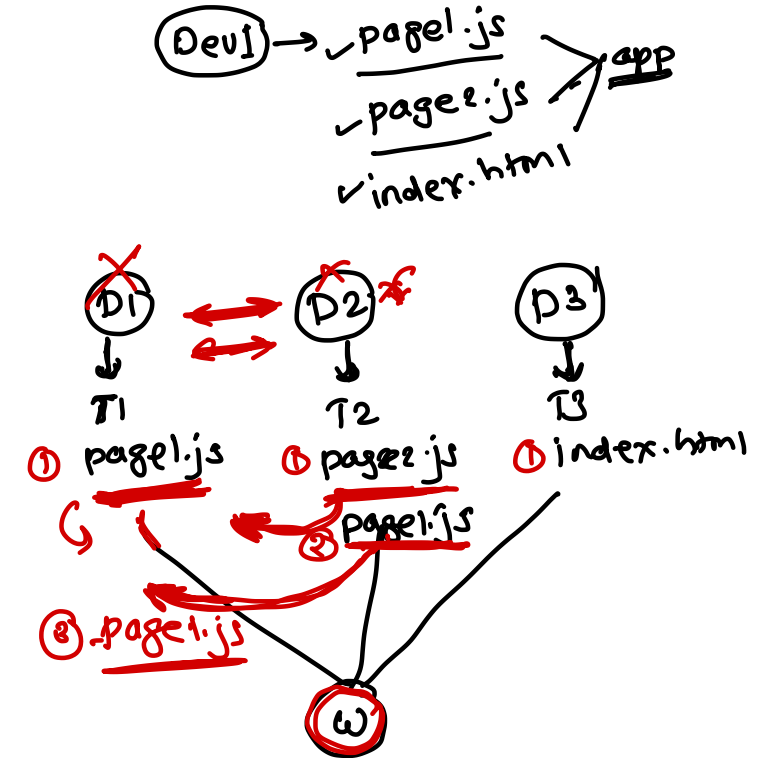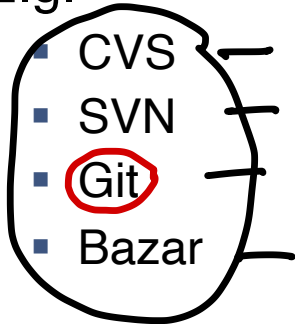# Source Code Management

Version Control System

- Also known as revision control or source code control system
- Is the management of documents (source code)
- Logical way to organize and control the revisions of source code
- Tracks and provides control over the changes made in the code
- E.g.
  - CVS
  - SVN
  - Git
  - Bazar

VCS Types – Client-Server

- Only one server maintains shared repository
- Every developer sends the changes to the same repository
- Disadvantages
    - not scalable
    - dependency on the server
- E.g.
    - Open source
        - CVS (Concurrent Version System)
        - SVN (Subversion)
    - Proprietary
        - AccuRev
        - Razor
        - TeamCity
   - Vault
        - Visual SourceSafe

VCS Types – Distributed

- Takes peer-to-peer approach to version control

- Synchronizes repositories by exchanging patches from peer to peer

- There is no single server which maintains the code, rather user has a working copy and full change history

- Disadvantages
  - Allows users to work productively even when not connected to internet
  - Common operations like commit, version history etc. are faster because there is no need to communicate with server
  - Communication with server is necessary only when developer wants to share the changes with others
  - Allows private work, users don't need to publish the changes for early drafts
  - Working copies function effectively as backups
  - Permits centralized control of the release version of code

Client-Server Vs Distributed
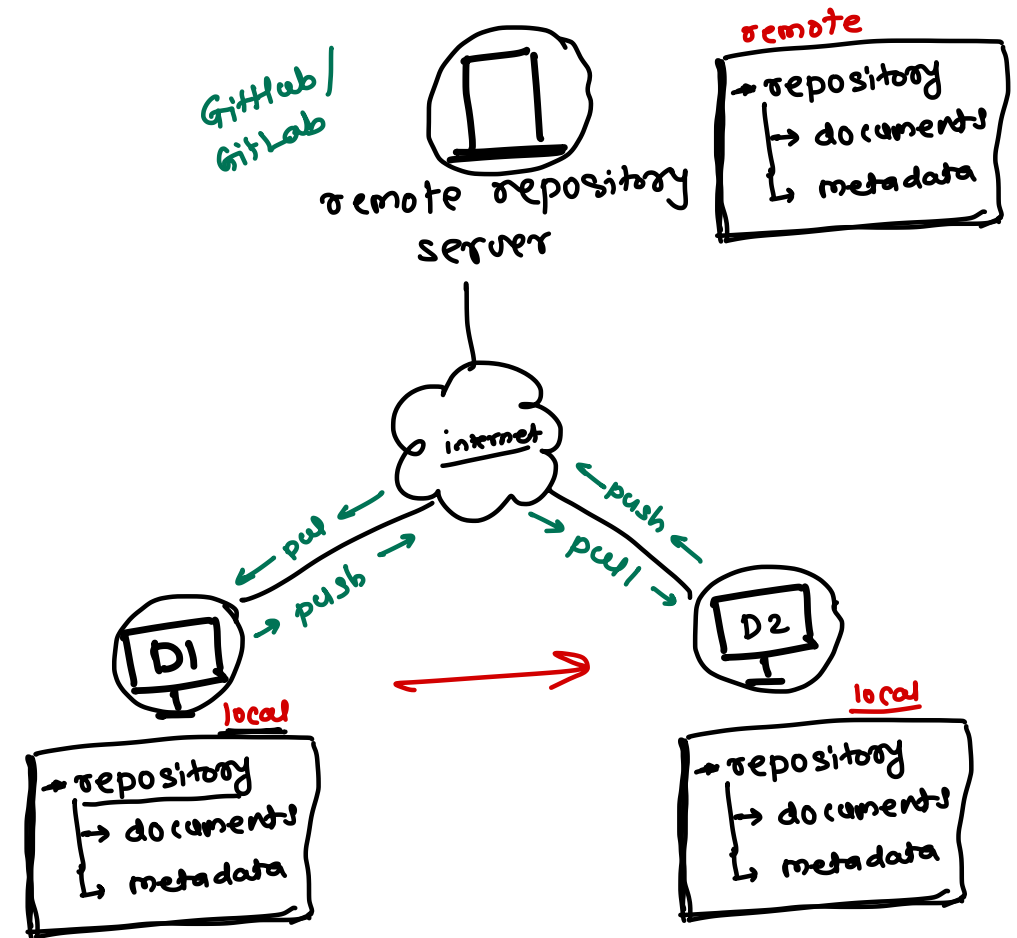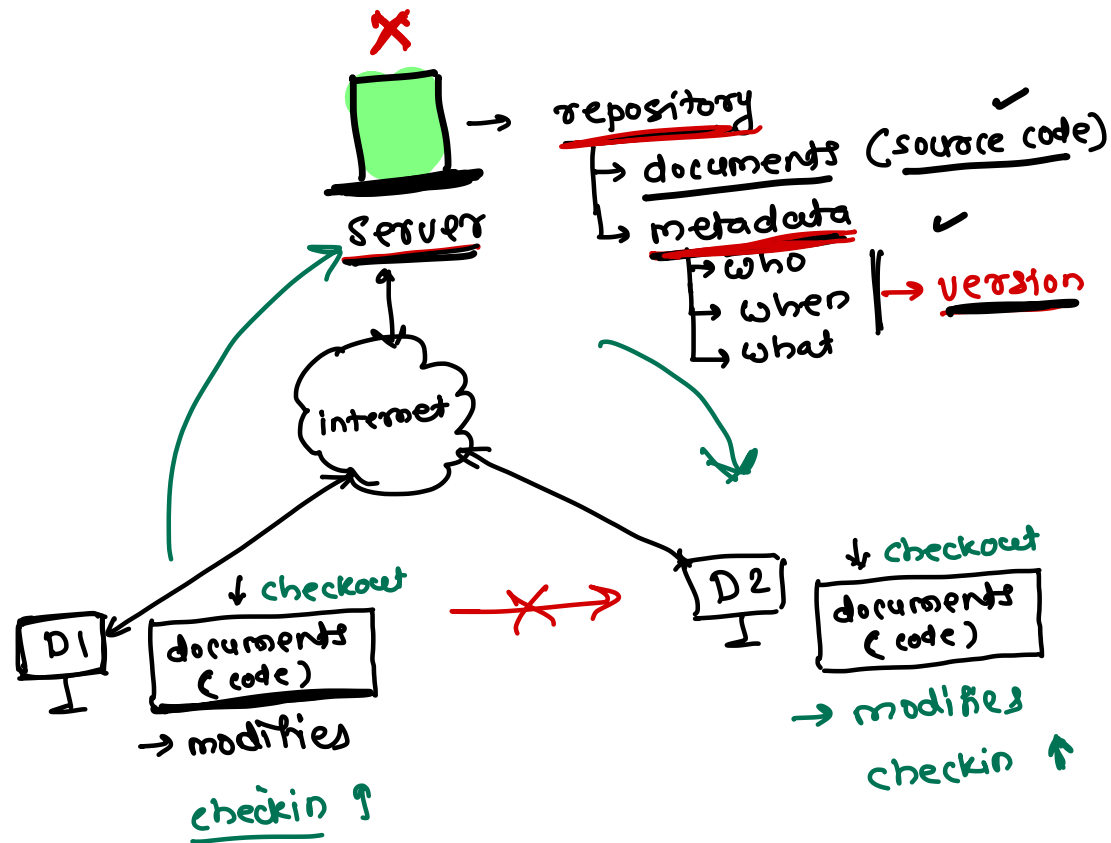
Git

Overview

- Git is a distributed revision control and source code management system

- Git was initially designed and developed by Linus Torvalds for Linux kernel development

- Git is a free software distributed under the terms of the GNU General Public License version 2

History

- The development of Git began on 3 April 2005

- Torvalds announced the project on 6 April

- It became self-hosting as of 7 April

- The first merge of multiple branches took place on 18 April

- Torvalds achieved his performance goals on 29 April

- On 16 June Git managed the kernel 2.6.12 release

- Torvalds turned over maintenance on 26 July 2005 to Junio Hamano, a major contributor to the project

Characteristics

- Strong support for non-linear development
- Distributed development
- Compatibility with existent systems and protocols
- Efficient handling of large projects
- Cryptographic authentication of history
- Toolkit-based design
- Pluggable merge strategies

Advantages

- Free and open source

- Fast and small

- Implicit backup

- Security

- No need of powerful hardware

- Easier branching

-

Terminologies

- Repository
  - Directory containing .git folder
- Object
  - Collection of key-value pairs
- Blobs (**B**inary **L**arge **Ob**ject)
  - Each version of a file is represented by blob
  - A blob holds the file data but doesn't contain any metadata about the file
  - It is a binary file, and in Git database, it is named as SHA1 hash of that file
  - In Git, files are not addressed by names. Everything is content-addressed
- Clone
  - Clone operation creates the instance of the repository
  - Clone operation not only checks out the working copy, but it also mirrors the complete repository
  - Users can perform many operations with this local repository
  - The only time networking gets involved is when the repository instances are being synchronized

Terminologies

- Pull
  - Pull operation copies the changes from a remote repository instance to a local
  - The pull operation is used for synchronization between two repository instances

- Push
  - Push operation copies changes from a local repository instance to a remote
  - This is used to store the changes permanently into the Git repository

- HEAD
  - HEAD is a pointer, which always points to the latest commit in the branch
  - Whenever you make a commit, HEAD is updated with the latest commit
  - The heads of the branches are stored in **.git/refs/heads/** directory

Terminologies

- Commits
    - Commit holds the current state of the repository.
    - A commit is also named by **SHA1** hash
    - A commit object as a node of the linked list
    - Every commit object has a pointer to the parent commit object
    - From a given commit, you can traverse back by looking at the parent pointer to view the history of the commit
- Branches
    - Branches are used to create another line of development
    - By default, Git has a master branch
    - Usually, a branch is created to work on a new feature
    - Once the feature is completed, it is merged back with the master branch and we delete the branch
    - Every branch is referenced by HEAD, which points to the latest commit in the branch
    - Whenever you make a commit, HEAD is updated with the latest commit

Life Cycle
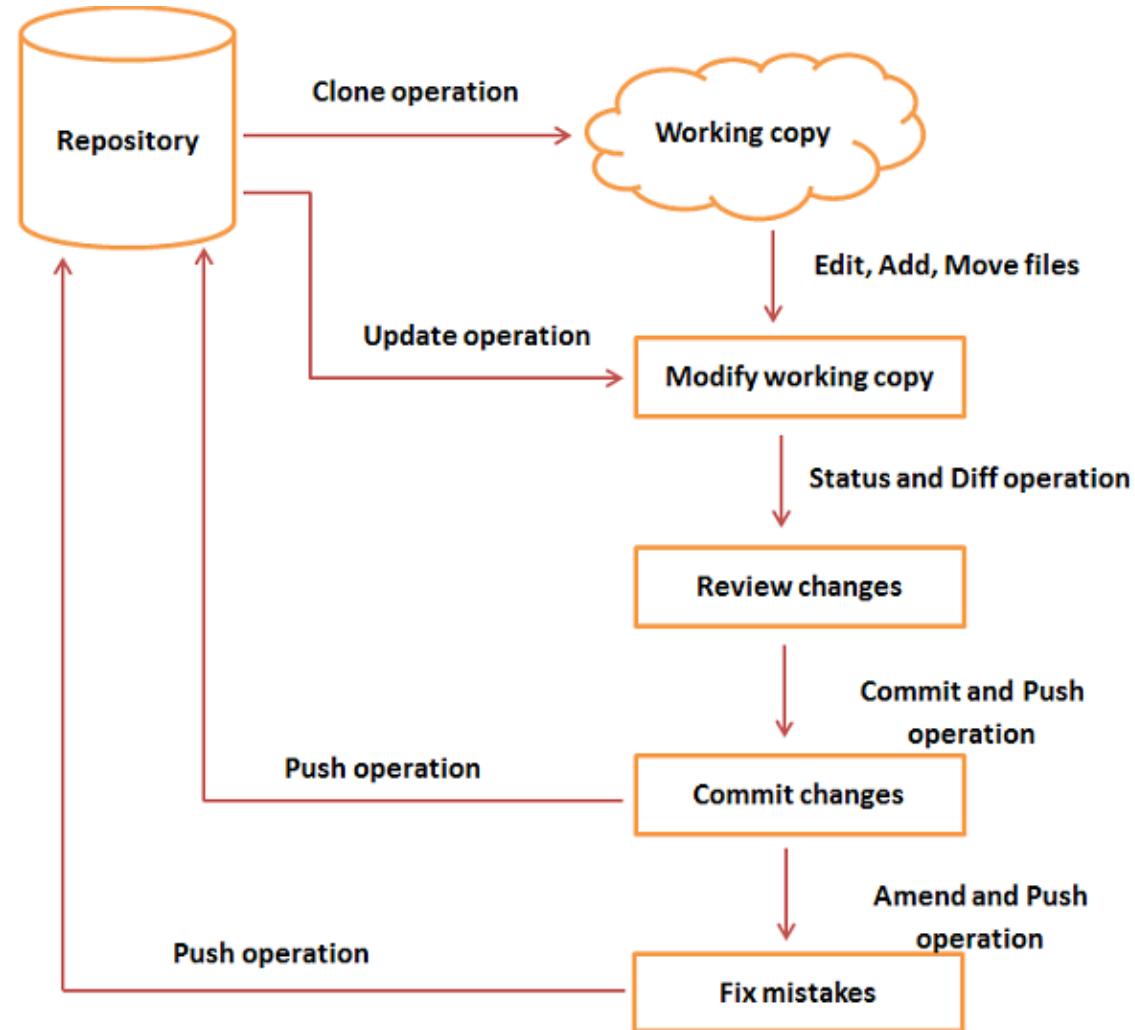
Installation and first time setup

- **Install git on ubuntu**

> sudo apt-get install git

- **List the global settings**

> git config --global --list

- **Setup global properties**

> git config --global user.name <user name>

> git config --global user.email <user email>

> git config --global core.editor <editor>

> git config –global merge.tool vimdiff

Basic Commands

- **Initialize a repository**

   > git init

- **Checking status**

   > git status

- **Adding files to commit**

   > git add .

- **Committing the changes**

   > git commit –m '<log message>'

Basic Commands

- **Checking logs**

  > git log

- **Checking difference**

  > git diff

- **Moving item**

  > git mv <source> <destination>

Basic Commands

- **Rename item**

  > git mv <old> <new>

- **Delete Item**

  > git rm <item>

- **Remove unwanted changes**

  > git checkout file

Branch

- Allows another line of development

- A way to write code without affecting the rest of your team

- Generally used for feature development

- Once confirmed the feature is working you can merge the branch in the master branch and release the build to customers

Why it is required ?

- So that you can work independently

- There will not be any conflicts with main code

- You can keep unstable code separated from stable code

- You can manage different features keeping away the main line code and there wont be any impact of the features on the main code

Branch management commands

- **Create a branch**

  > git branch <branch name>

- **Checkout a branch**

  > git checkout <branch name>

- **Merge a branch**

  > git merge <branch name>

- **Delete a branch**

  > git branch –d <branch name>

GitHub

Overview

- GitHub is a web-based hosting service for version control using Git

- It provides access control and several collaboration features
  - bug tracking
  - feature requests
  - task management
  - wikis for every project

- Developer uses github for sharing repositories with other developers

Workflow

- Create a project on GitHub

- Clone repository on the local machine

- Add/modify code locally

- Commit the code locally

- Push the code to the GitHub repository

- Allow other developers to get the code by using git pull operations

Workflow commands

- **Add remote repository**

  > git remote add <name> <url>

- **Clone remote repository**

  > git clone <url>

- **Push the changes**

  > git push <name> <branch>

- **Pull the changes**

  > git pull