# Source Code Management

SCM / VCS

## Why do we need Version Control System?

- Many people's version-control method of choice is to copy files into another directory
- This approach is very common because it is so simple
- But it is also incredibly error prone
- It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to
- To deal with this issue, programmers long ago developed VCS

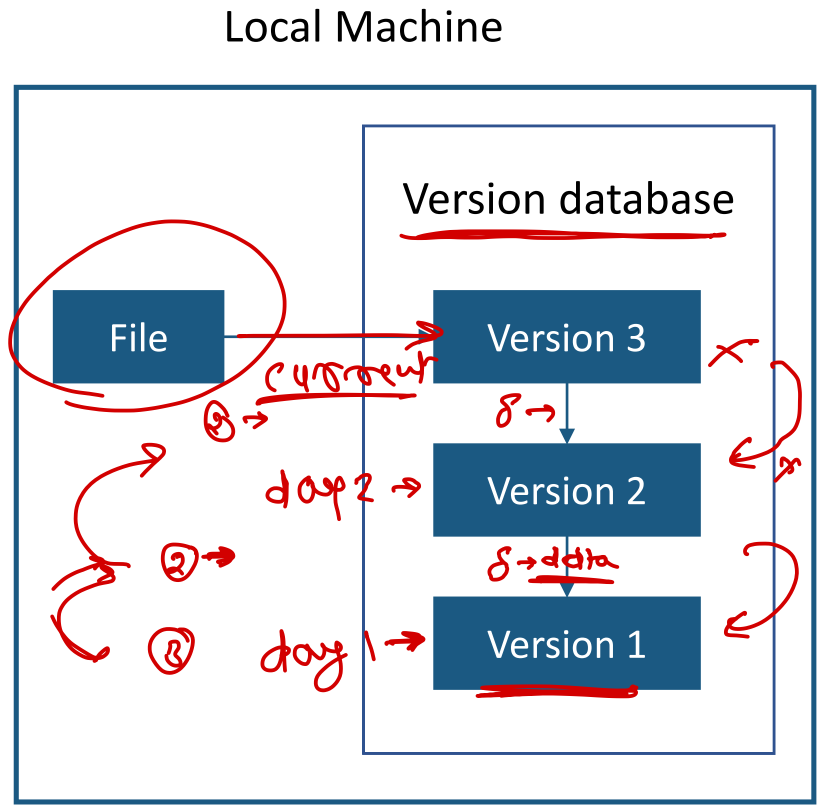
## What is Version Control System?

- System that records changes to a file(s) over time so that you can recall specific versions later
- It allows you
  - to revert files to a previous state
  - to revert the entire project to a previous state
  - to compare changes over time
  - to see who last modified something that might be causing a problem
  - to see who introduced an issue and when
- Using a VCS also generally means that if you screw things up or lose files, you can easily recover

## Local Version Control System

- Contains simple database that kept all the changes to files under revision control
- One of the more popular VCS tools was a system called RCS
- RCS works by keeping patch sets (that is, the differences between files) in a special format on disk
- It can then re-create what any file looked like at any point in time by adding up all the patches

update
→ add
→ modify
→ delete

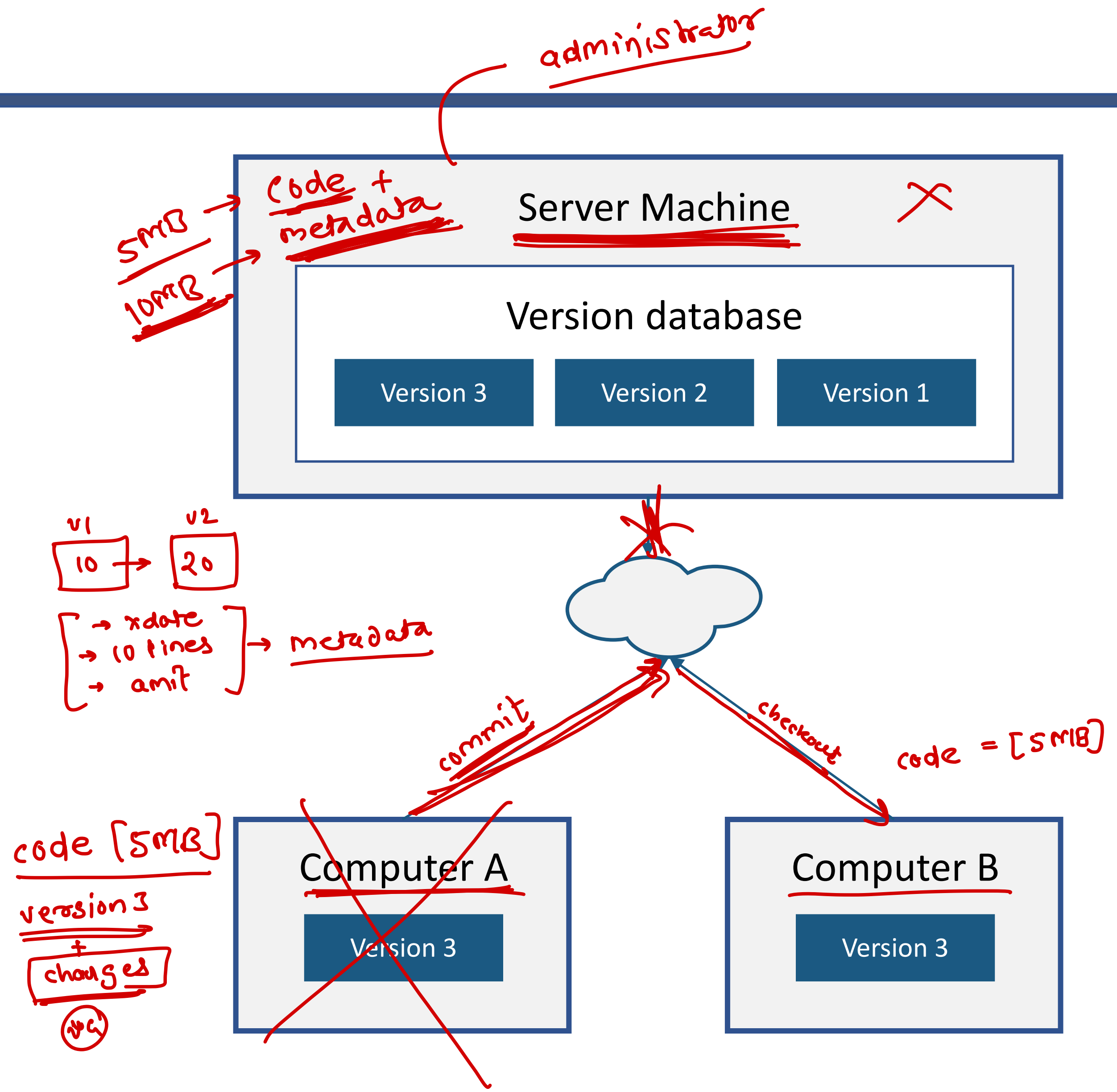


# Centralized Version Control Systems

- People need to collaborate with developers on other systems
- To deal with this problem, Centralized Version Control Systems (CVCSs) were developed
- These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place
- E.g. CVS, Subversion, and Perforce

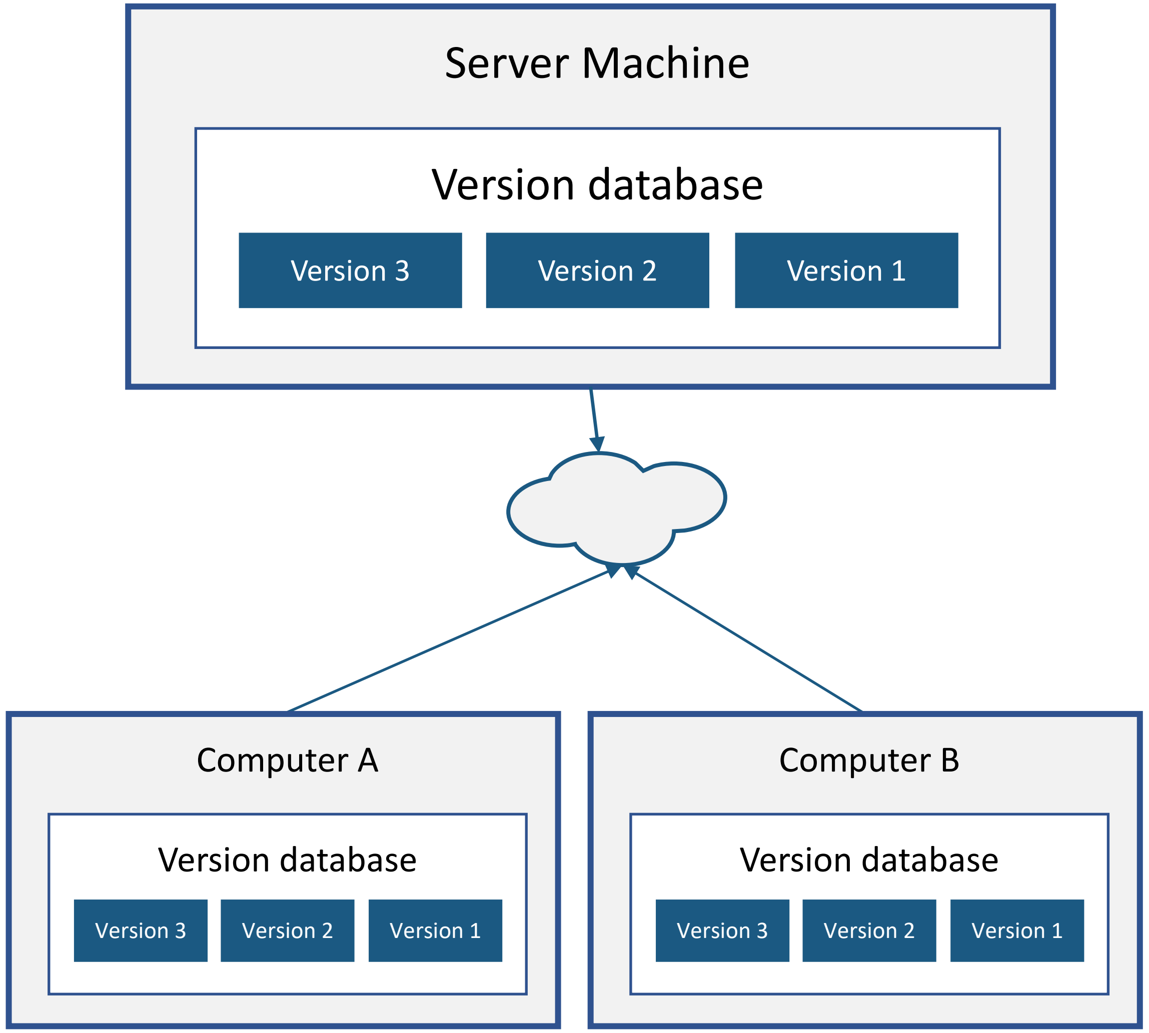
(SVN)

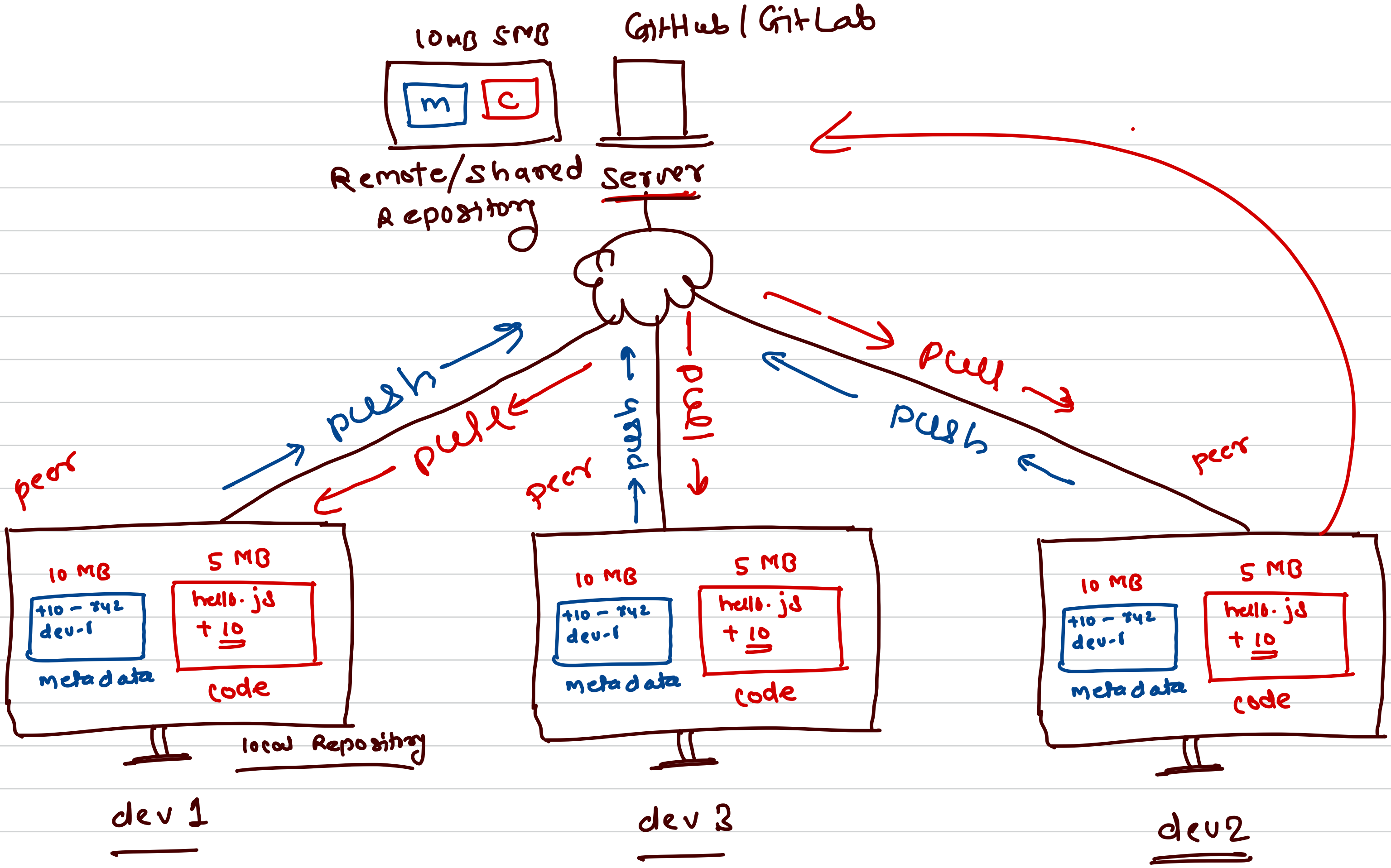
svn diff <:>



## Distributed Version Control Systems

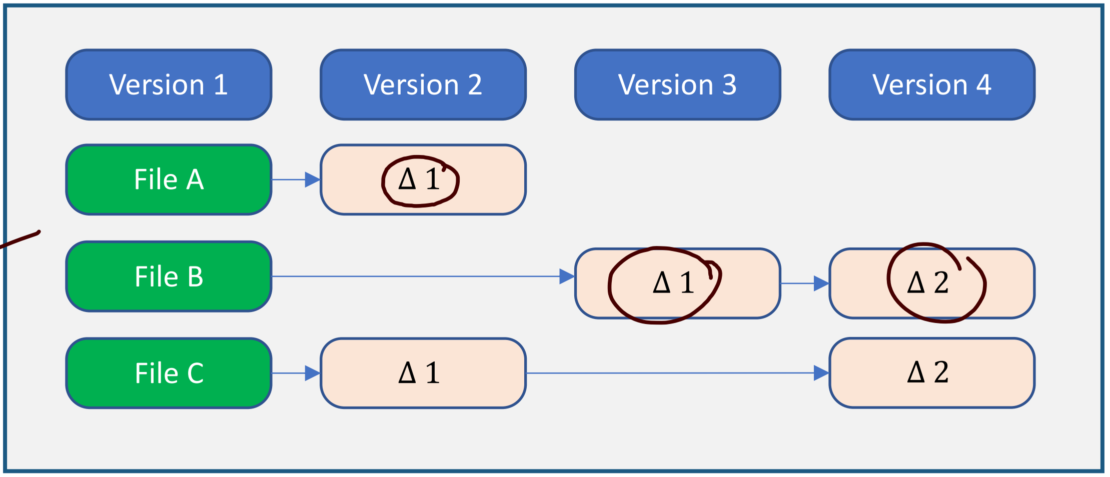
- Clients don't just check out the latest snapshot of the files, rather they fully mirror the repository
- Thus if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it
- Every checkout is really a full backup of all the data
- Furthermore, many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project

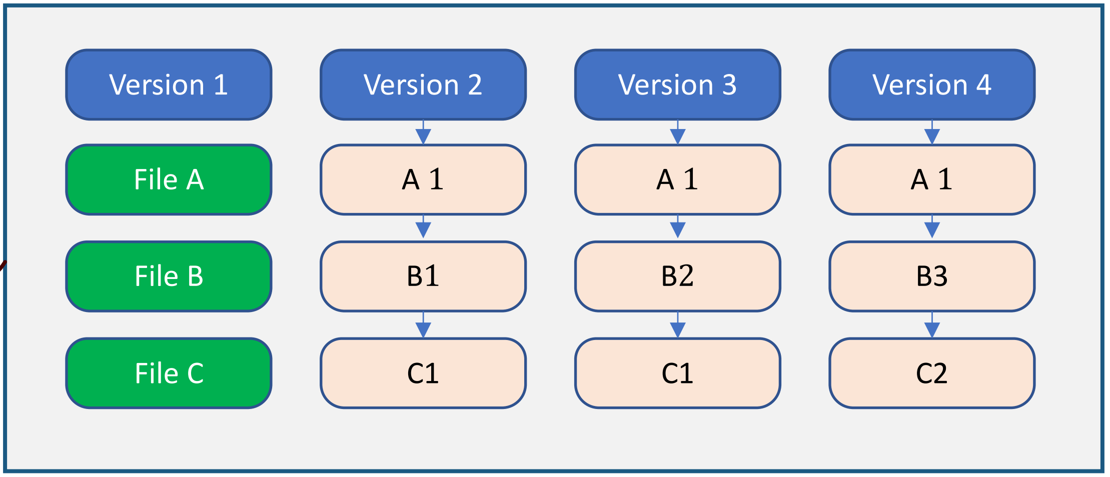



10MB 5MB
m
c
Remote/shared Repository
GitHub | GitLab
Server
push
pull
push
pull
pull
push
peer
peer
peer
10 MB
5 MB
+10 - 742
dev-1
hello.js
+ 10
metadata
code
local Repository
dev 1
dev 3
dev2

## What is Git?

- Git is one of the distributed version control systems
- The major difference between Git and any other VCS is the way Git thinks about its data
- Unlike other VCS tools, Git uses snapshots and not the differences
- Others think of the information they keep as a set of files and the changes made to each file over time
- Git thinks of its data more like a set of snapshots of a miniature filesystem
- Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot
- To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored





## 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

## A little bit history about Git

- The Linux kernel is an open source software project of very large scope
- From 1991–2002, changes to the software were passed around as patches and archived files
- In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper
- In 2005, the relationship with BitKeeper broken down and tool's free-of-charge status was revoked
- tool's free-of-charge status was revoked (and in particular Linus Torvalds) to develop their own tool based on some of the lessons they learned while using BitKeeper
- Some of the goals of the new system were
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)

## Characteristics

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

## Advantages

- Free and open source
- Fast and small
- Implicit backup
- Security
- No need of powerful hardware
- Easier branching

## Installation and first time setup

- **Install git on ubuntu**

```
> sudo apt-get install git
```

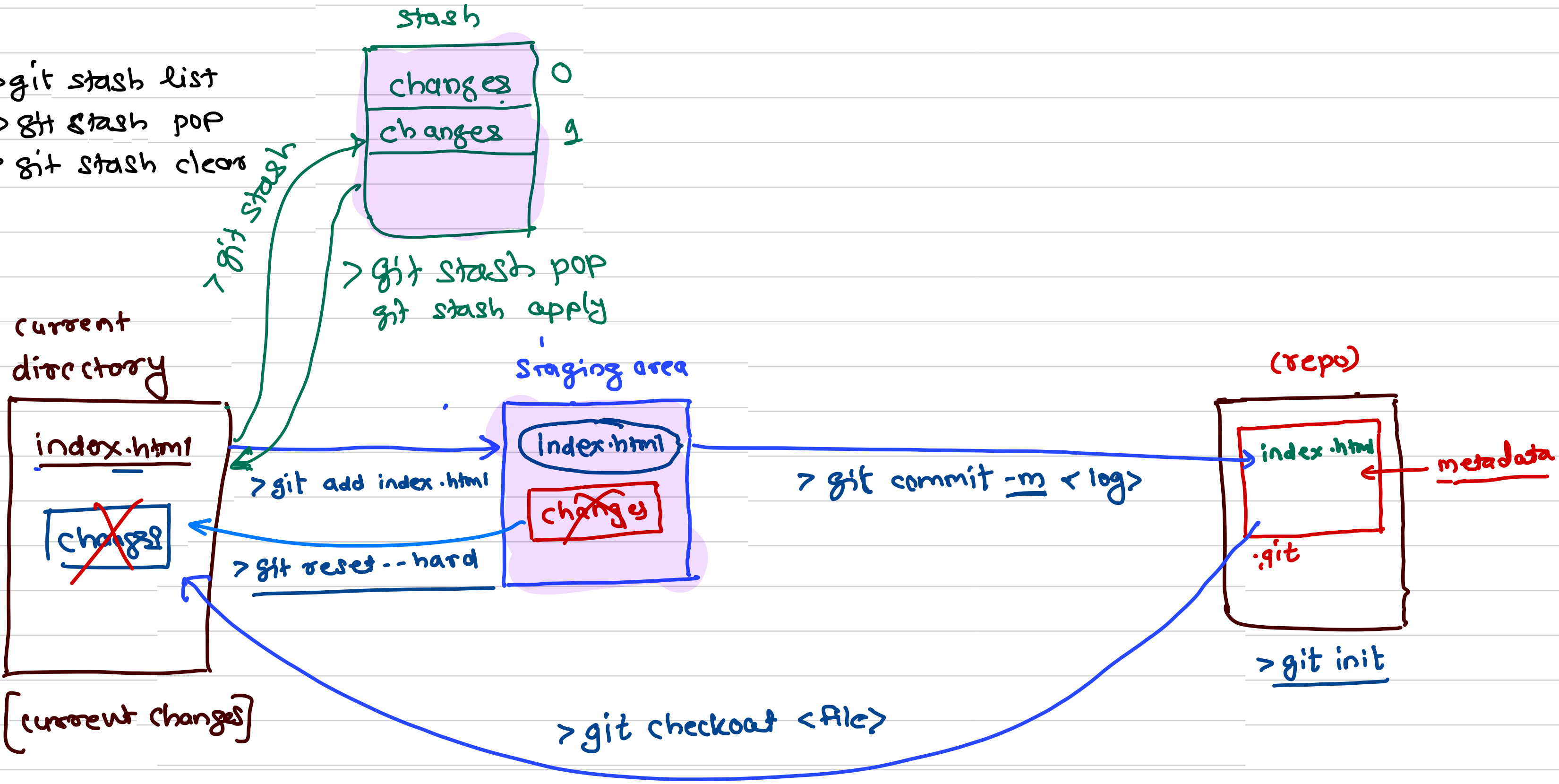
- **List the global settings**

```
> git config --global --list
```

- **Setup global properties**

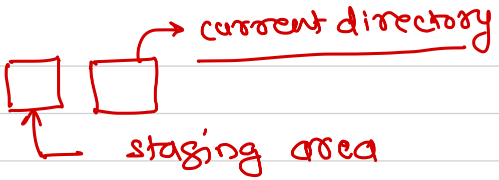
```
> git config --global user.name <user name>
> git config --global user.email <user email>
> git config --global core.editor <editor>
> git config –global merge.tool vimdiff
```

```
>git stash list
>git stash pop
>git stash clear
```



```
>git stash
>git stash pop
git stash apply
>git add index.html
>git reset --hard
>git commit -m <log>
>git init
>git checkout <file>
```

## Status



$\boxed{?}\ \boxed{?}$ → untracked

$\boxed{A}\ \boxed{\phantom{A}}$ → added to the stage area

$\boxed{\phantom{M}}\ \boxed{M}$ → modified in current directory

$\boxed{M}\ \boxed{\phantom{M}}$ → modified and added to stage area

$\boxed{U}\ \boxed{U}$ → file has conflicts

## 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>
```

## 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

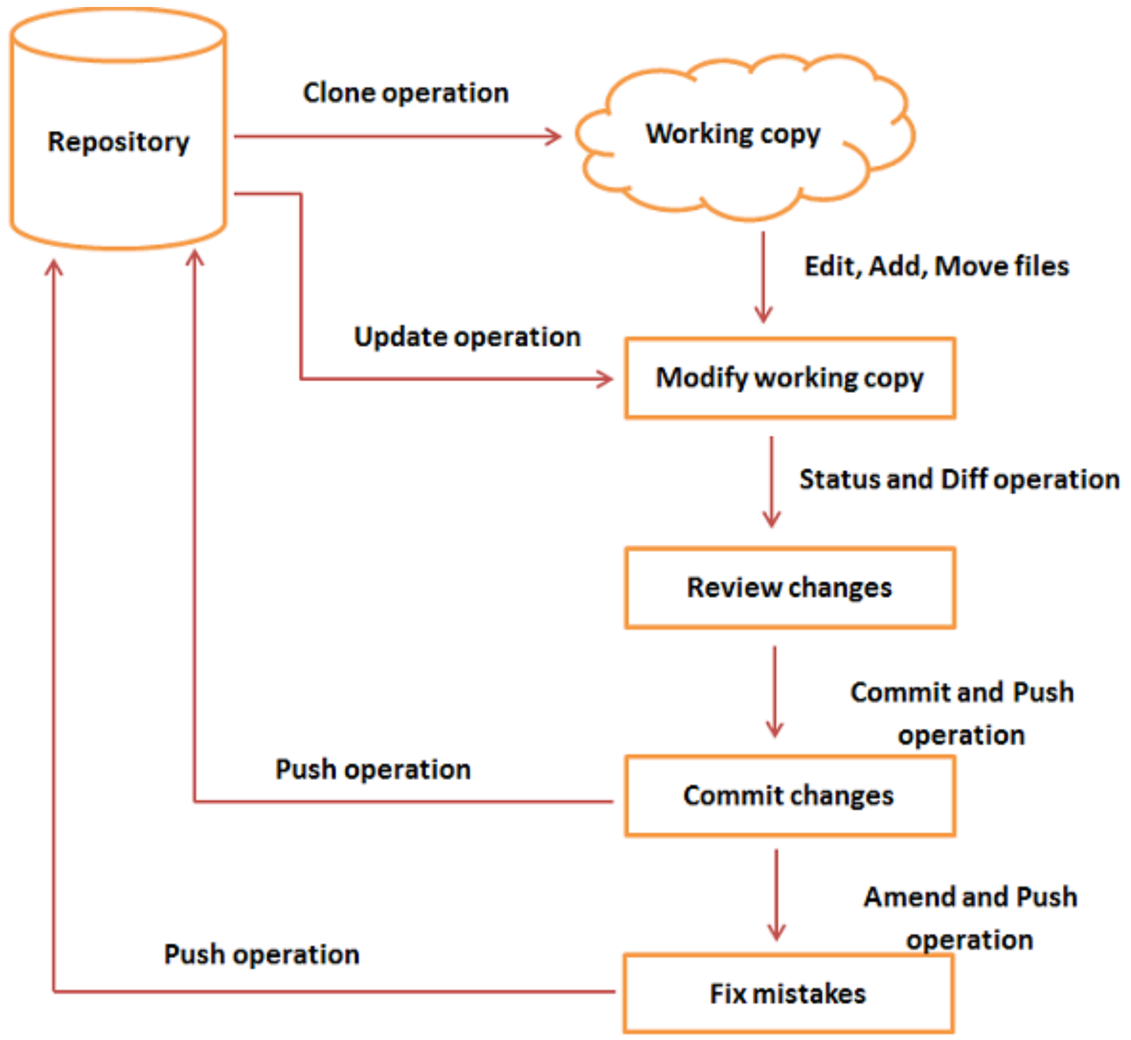
## Terminologies

- 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


Repository
Clone operation
Working copy
Edit, Add, Move files
Update operation
Modify working copy
Status and Diff operation
Review changes
Commit and Push operation
Push operation
Commit changes
Amend and Push operation
Push operation
Fix mistakes

## 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**

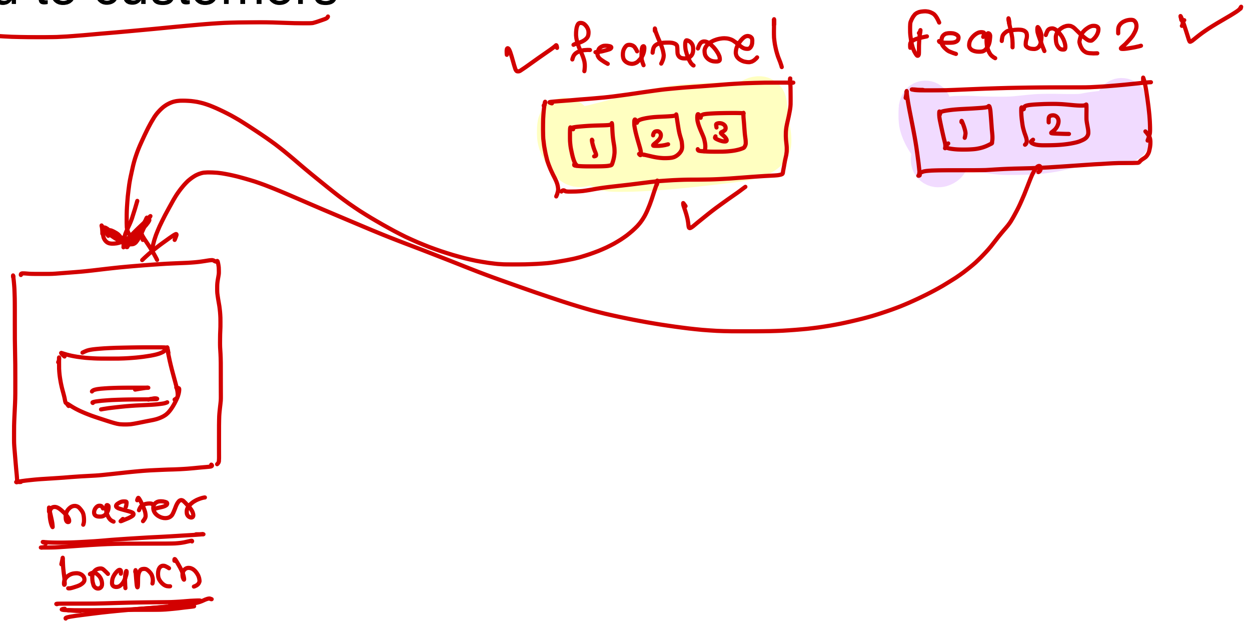
```
> git rm <item>
```

- **Remove unwanted changes**

```
> git checkout file
```

## Branch → master

- 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

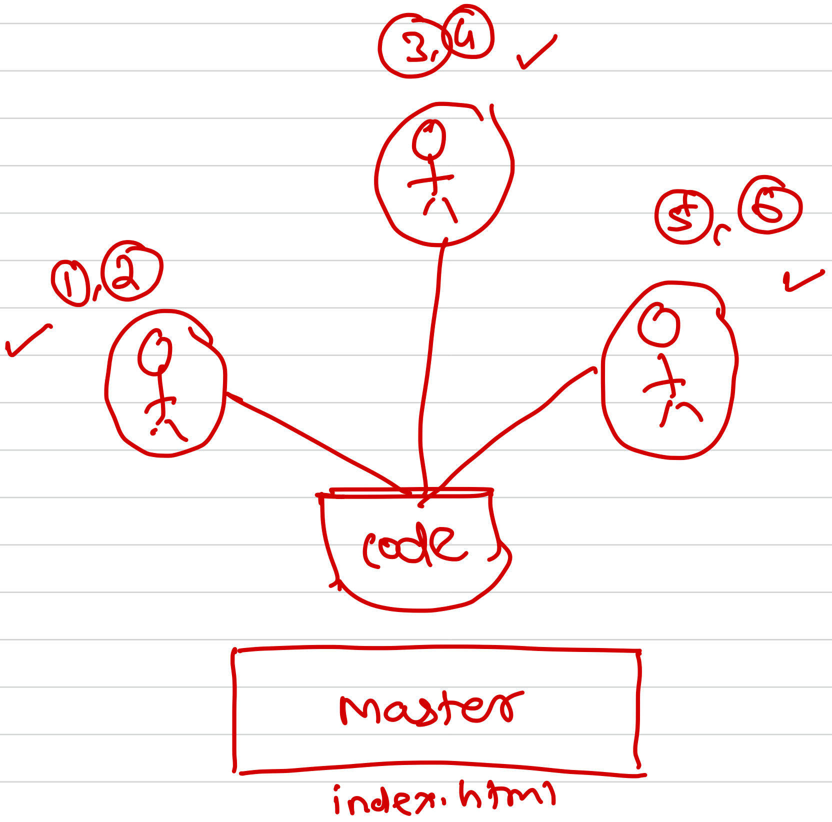


master

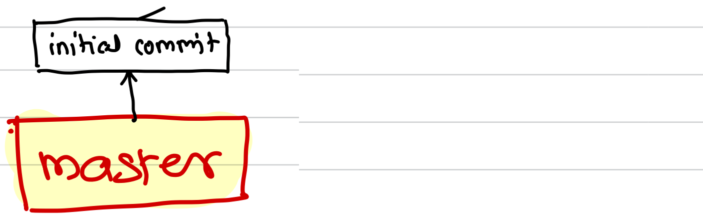
① latest changes
② stable
③ non-buggy ?
④ non-crashing

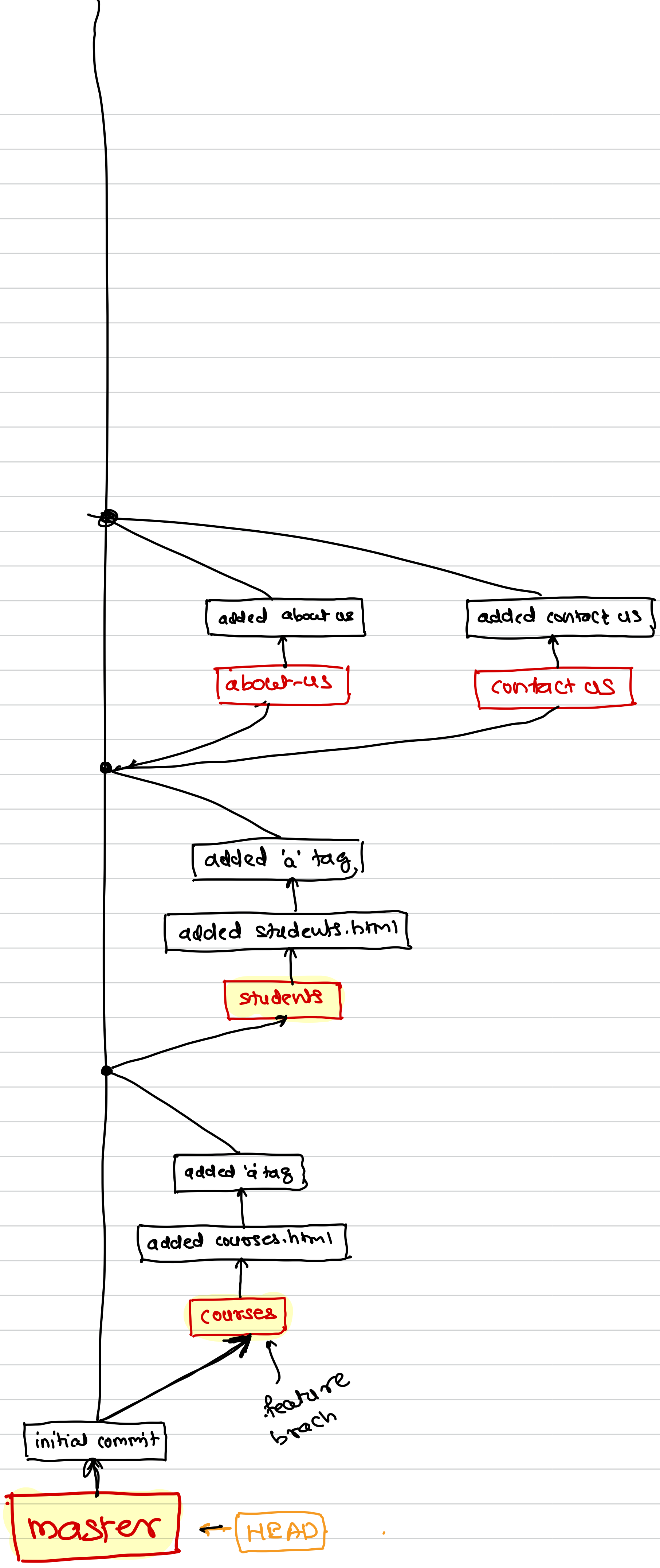
## Sunbeam website - Features

1. ✓ courses
2. ✓ students
3. ✓ faculties
4. ✓ feedbacks
5. ✓ about.us
6. ✓ contact-us



```
> git branch courses
> git brach
> git checkout courses
```




added about us
about-us
added contact us
contact us
added 'a' tag
added students.html
students
added 'a' tag
added courses.html
courses
feature branch
initial commit
master
HEAD

## Why is it 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>
```

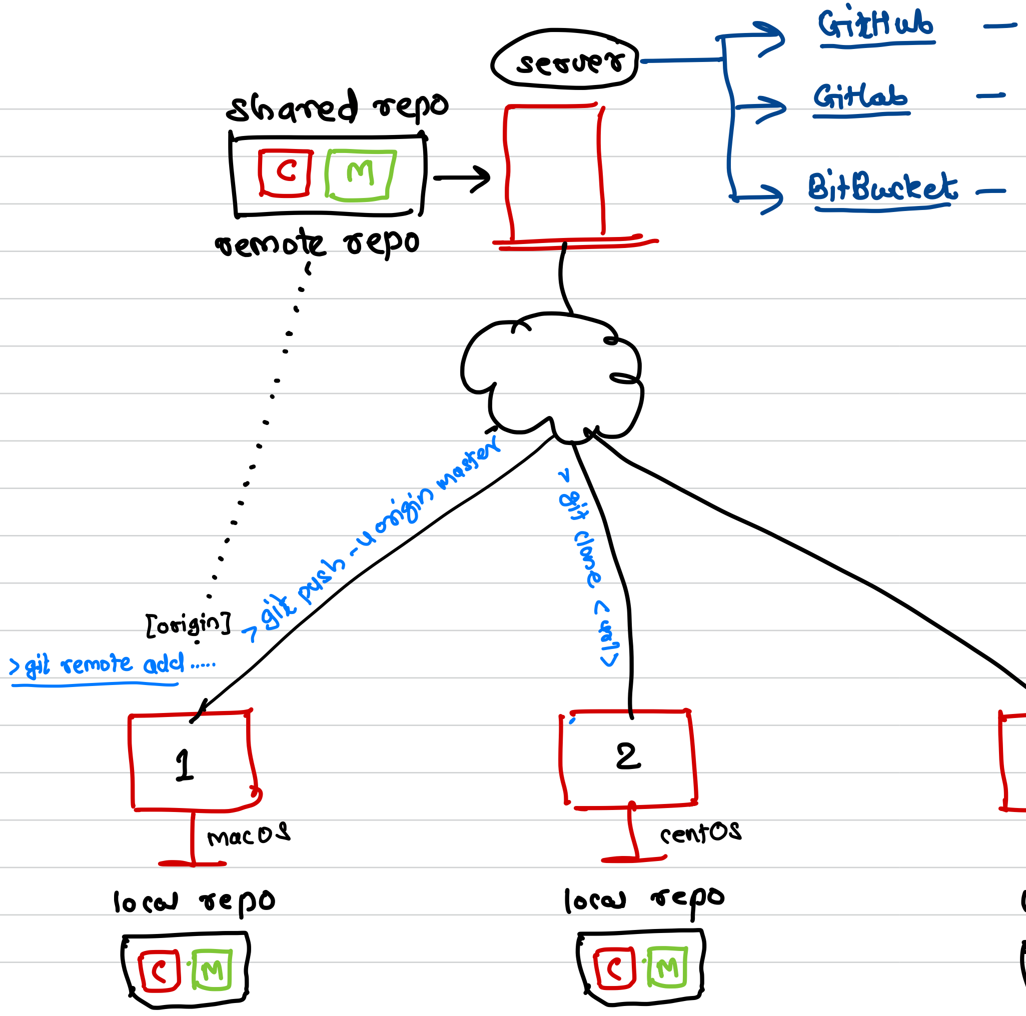
- **Merge a branch**

```
> git merge <branch name>
```

- **Delete a branch**

```
> git branch –d <branch name>
```

# $\boxed{\text{GitHub}}$



```
> git remote add .....
> git push -u origin master
> git clone <url>
```

## 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
```