Version Control Systems

Ganesh Palnitkar

Agenda

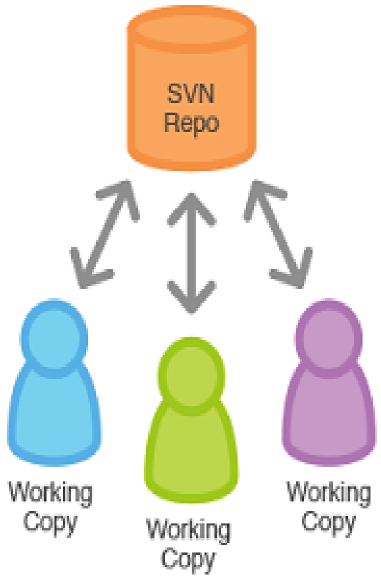
- Importance of Version Control System
- Types of Version Control Systems
  - Distributed VCS
  - Centralized VCS
- Importance of Branching and merging
- Branching and Merging strategies
- Learning GIT – Common commands

Agenda …cont.

- Introduction
- Version control systems
- Local, Centralized and distributed
- Installing Git on Linux, Windows
- Local repository Configuration.
- Git Essentials
- Cloning, check-in and committing
- Fetch pull and remote
- Branching and Merging. Merging Strategies.
- Git Commands
- Git remote repository application – GitHub, GitLab
- Working with GitHub as a remote repository

About Version Control Systems

- **Version control software** is an essential part of the every-day of the modern software team's professional practices.
  - Version control software keeps track of every modification to the code in a special kind of database.
  - Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.
  - Helps teams to solve problems like, tracking every individual change by each contributor and helping prevent concurrent work from conflict.
  - Conflicts if any should be solved using help of tool or manually.



Benefits:

- No need to keep multiple backups.
- Allows multiple people to work on same file / project.
- A complete long-term change history of every file.
- Branching and merging – maintain code as per projects / release / functionality etc.
- Traceability - ability to trace each change made to the software and connect it to project management and bug tracking software.
- Easily switch / work on earlier file versions.

Types of Version Control System

Centralized VCS:

Workflow:

- Pull down any changes other people have made from the central server.
- Make your changes, and make sure they work properly.
- Commit your changes to the central server, so other programmers can see them.

Benefits:

- Project history and database is maintained on centralized server, beneficial incase of large project size and history.

Disadvantages:

- Internet / network connection is must.
- Frequent (daily) database sync is necessary.
- Pull and push to centralized server can be time taking.

e.g. : SVN, TFS, Perforce

Distributed VCS

Workflow:

- Clone entire repository on local machine.
- Maintain all code changes on local machine.
- Incase of sharing the code with others, merge the code on central GIT repo.

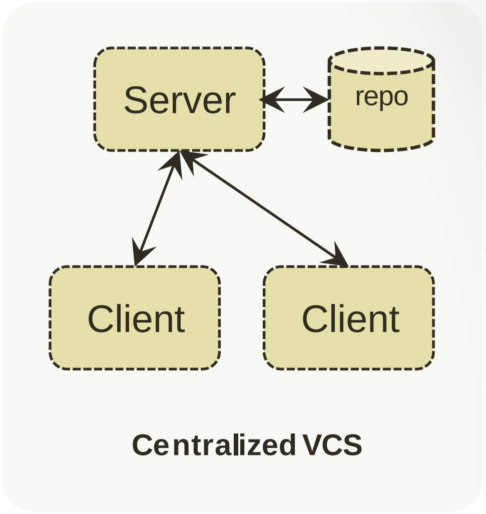
Benefits:

- Extremely fast, as database resides on local drive.
- Committing to a changeset can be done locally. Group of changesets then can be pushed to remote repository.
- No need of internet connection.

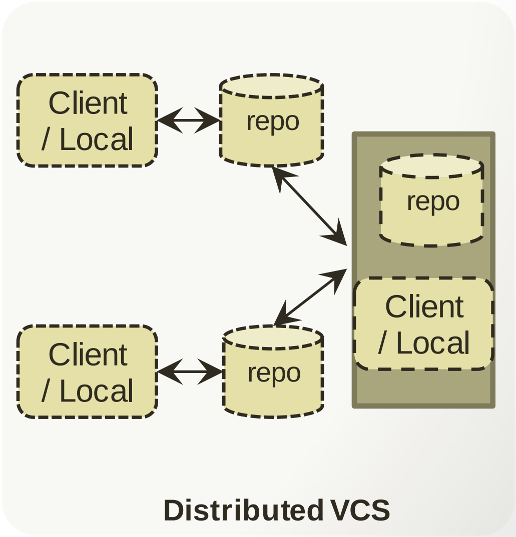
Disadvantages:

- The only major one can be because of large project history / size. Initial cloning and maintaining huge database can eat up space on local machine.
- Database backup should be done locally.

e.g. GIT, Mercurial

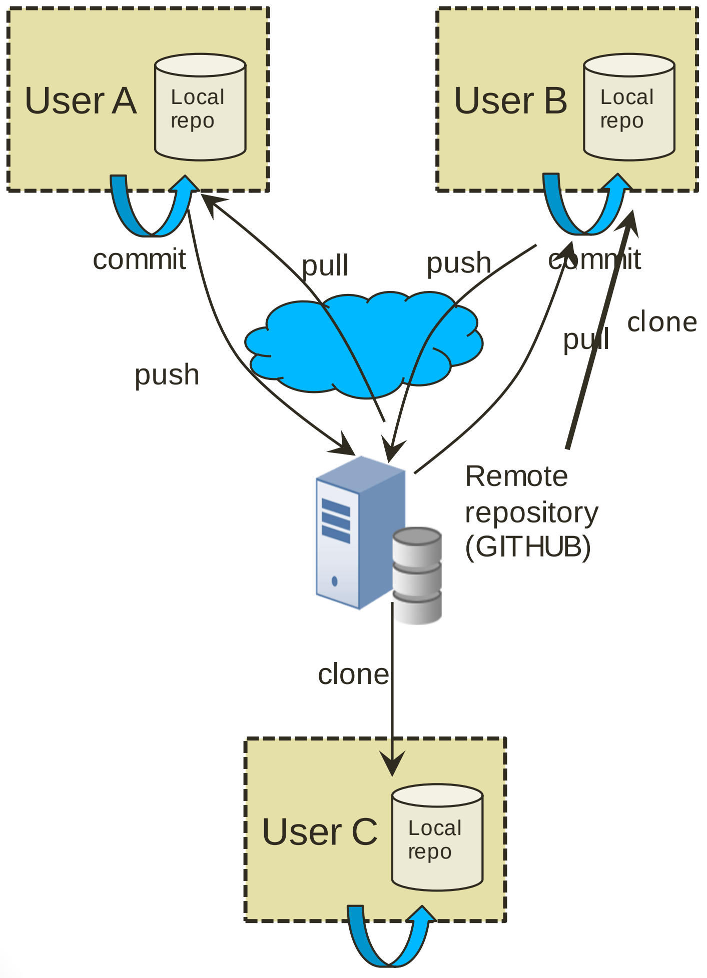


Centralized VCS



Distributed VCS

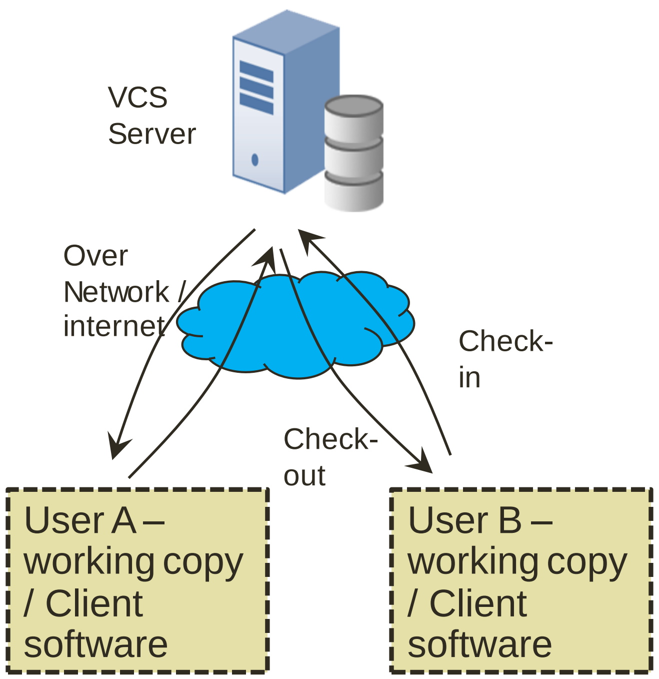
Distributed VCS



DVCS

- **User A** as the first user / Developer in the team to start coding.
- **User A** creates a **local repository** on his local machine / development environment. Local version control operations
- **User B** joins the team to speed up the development work. He asks for a base code copy from User A.
- **User A** creates a **remote repository** on GitHub / GitLab and **Push** the code to remote repo.
- **User B** makes a clone of remote repo, a particular ranch to his / her local machine and gets a local repo created.
- **Later User C** is asked to work on a new feature.
- **User C** makes a **clone** of remote repo, a particular ranch to his / her local machine and gets a local repo created.

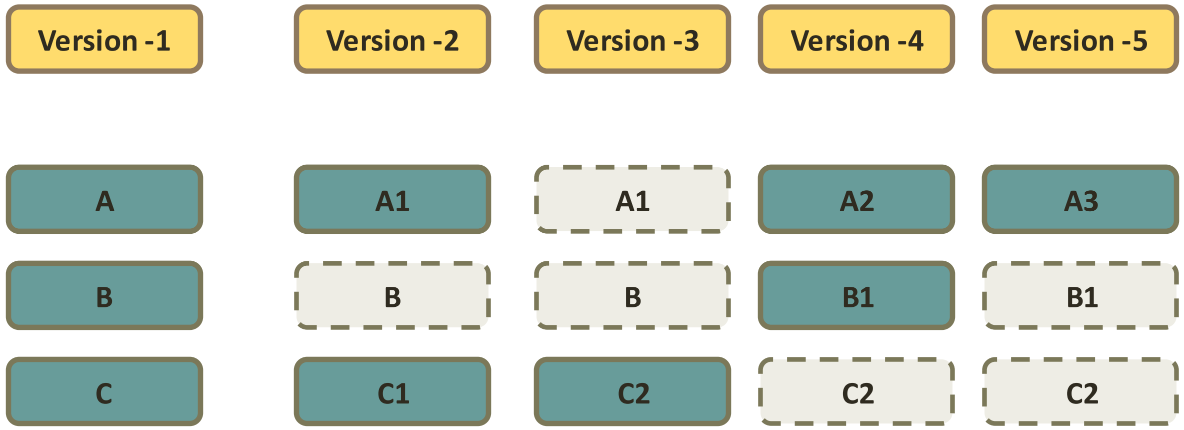
Centralized VCS


VCS Server
Over Network / internet
Check-in
Check-out
User A – working copy / Client software
User B – working copy / Client software

CVCS

- A centralized repository on a central server (SVN) is created.
- Developers / users will make a pull / checkout from the central repository.
- Only latest version is pull / checked out in to local workspace on the developer's machine.
- Users / Developers will continue to check-in and check-out from central server to local workspace

GIT versioning using snapshots



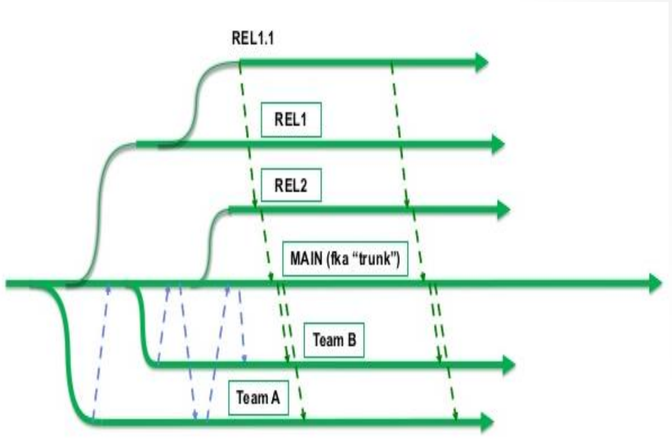
Branching and Merging strategies

- Brach by release, “staircase model”.
- Branch by feature. Each distinct branch.
- Hotfix branch.

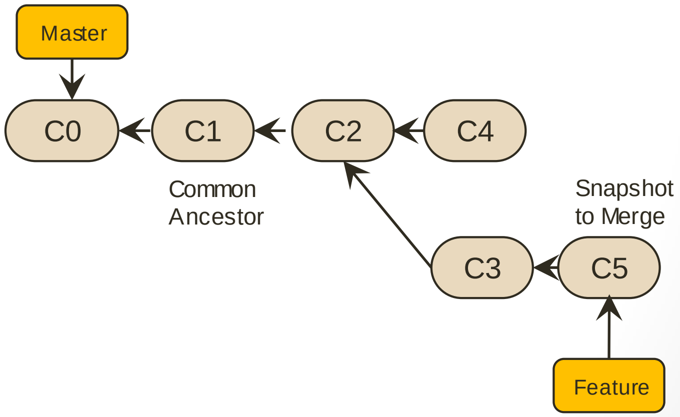
Best practices:

- Keep option of branching only if required, keep it simple.
- Incase of centralized VCS, merging should be done frequently / sooner (daily recommended)
- Use tagging as and when required.
- Delete unwanted branches.
- Identify shared components in details and in advance.

- **Merge only when your branch is compliable and stable.**



Branching and merging scenario

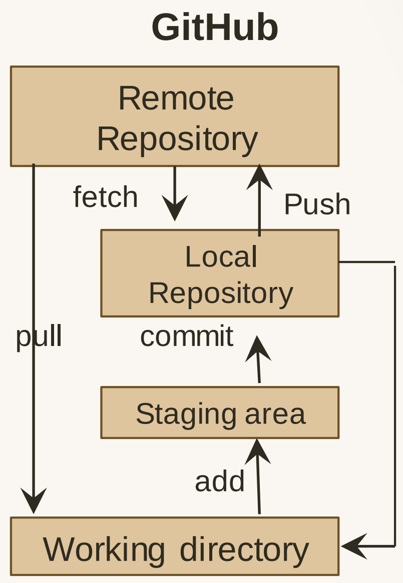


Simple Branching and merging in GIT project

GIT - DVCS

- GIT has bash /command-line interface and GUI as well.
- Git is a distributed VCS. Provides extremely fast operation. Does not need centralized server. Once the entire repository is pulled on local machine, network / internet connection is not required for most of the VCS operations.

- GIT works on most of the OS platforms (OSx, Windows, Unix / Linux).
- On installation, the GIT config should be run to configure user name, email ID, etc.
- In order to start using the VCS, the folder that you want to version control, run the command 'git init'. This enables the VCS and tracking of file changes in the folder.

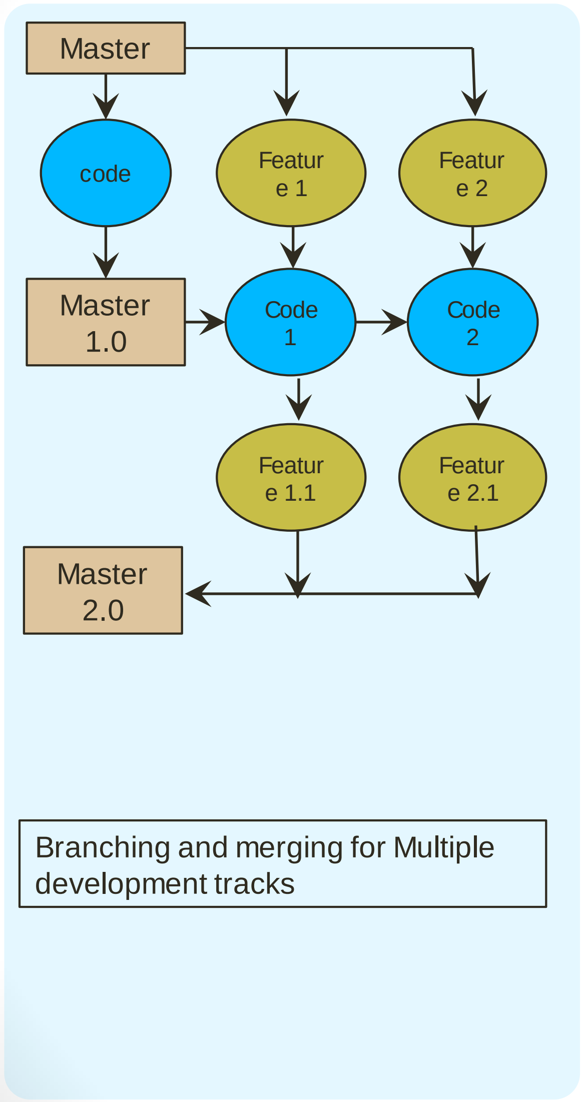


Local GIT repository

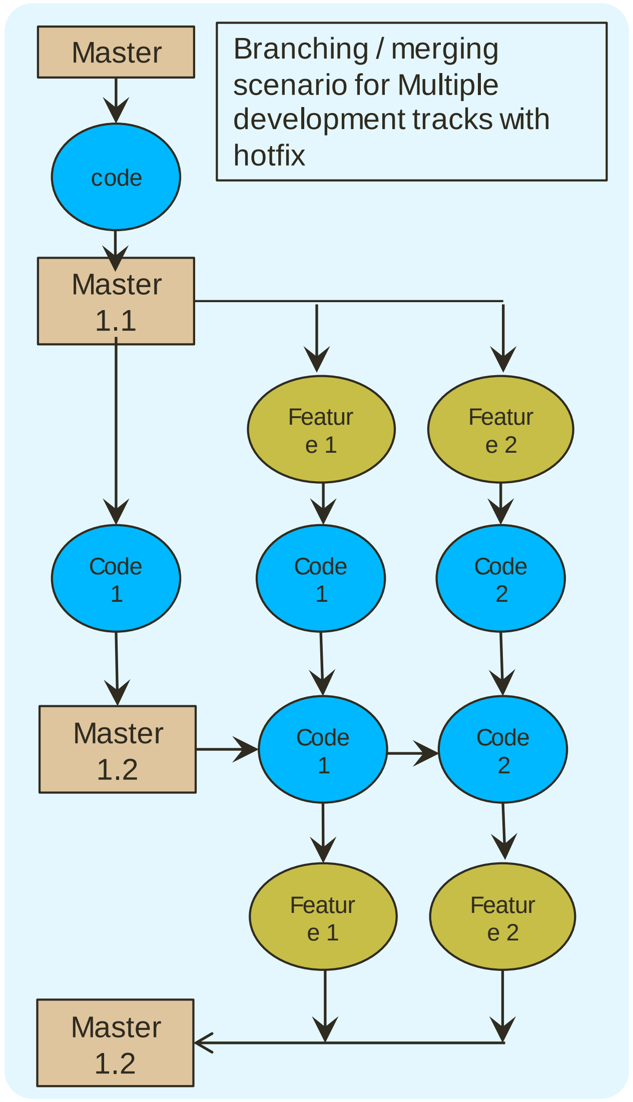
Download 'git' from, http://git-scm.com

- **Working area:** similar to work area, development environment.
- **Staging area:** where you store a place a snapshot before committing changes to the local repo.
- **Local repo:** is a copy of remote repository. This is where you have all versioning of data/code/artifacts maintained.
- **Remote repository:** this can be the GitHub repository or a remote repository maintained on a server on intranet.

GIT – Branching Strategies



Branching and merging for Multiple development tracks



Branching / merging scenario for Multiple development tracks with hotfix

To rename a remote repository,

```
git remote rename <remote-name><URL>
```

Verify after renaming,

```
git remote –v
```

To delete existing remote,

```
git remote rm <url>
git remote –v
```

To push commits made to local repo onto remote repo,

```
git push origin master
git push <remote name> <branch>
```

To push local branch to remote branch and rename remote,

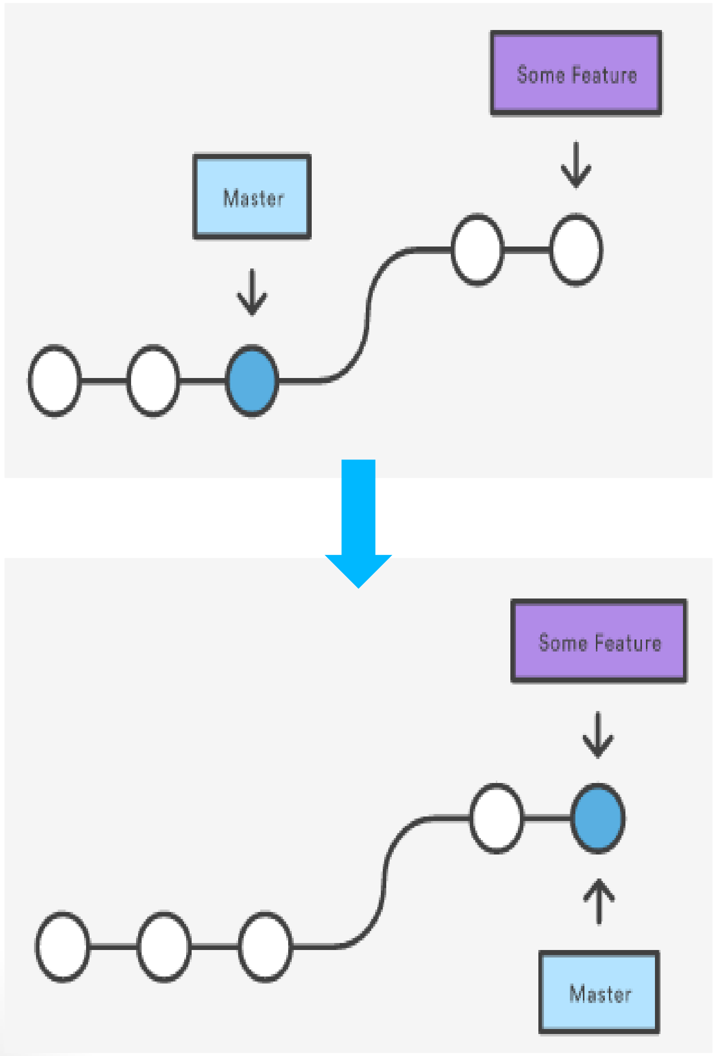
```
git push <remote-name><local branch>:<remote branch-name>
```

GIT common Commands

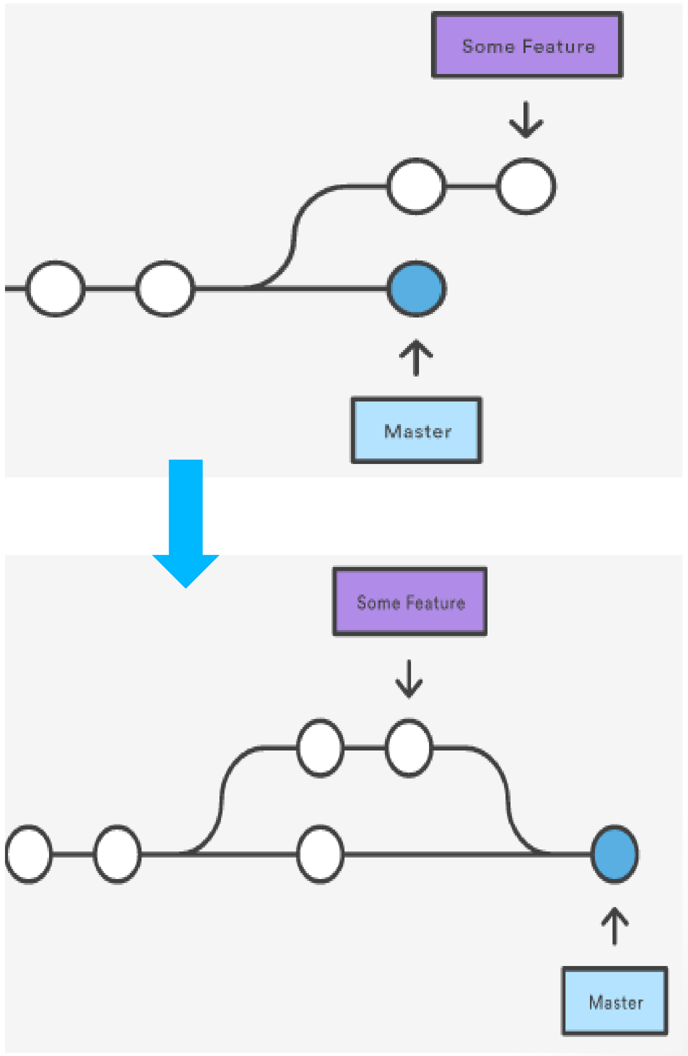
- Below are certain commands that are used often while we use ‘GIT’.
  - git status → provides status of changed, created, deleted files objects
  - git add → indexing done for selected or all files and added to staging area
  - git rm –cached → to remove file added to staging area
  - git commit –m “comment” → final commit of object to local repo database
  - git log --graph -p → use ‘shift+z’ to come out of the log
  - git diff → provides info about what has changed in the file
  - git diff --cached → diff for the files in the staging area.
  - git branch → provides list of all branches
  - git branch <branch name> → to create a branch
  - git remote add <name> <url> → to add remote repo ref.
  - git checkout <branch name> → switch to mentioned branch
  - git clone <https://remote repo> → clone from remote repo.
  - git pull <repo name> <branch name> → pull updates for defined branch from remote repo
- Always perform a ‘Pull’ action before ‘pushing’ the code to remote repo.

GIT Merge operations

Fast-forward Merge



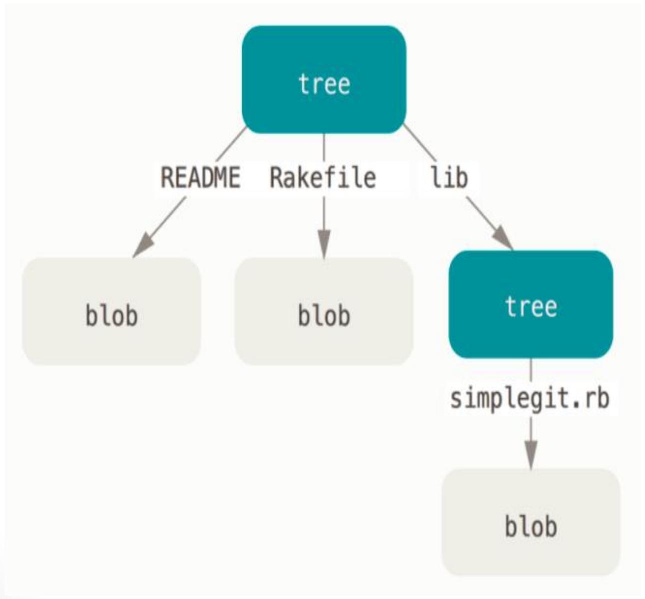
3-way or Recursive Merge

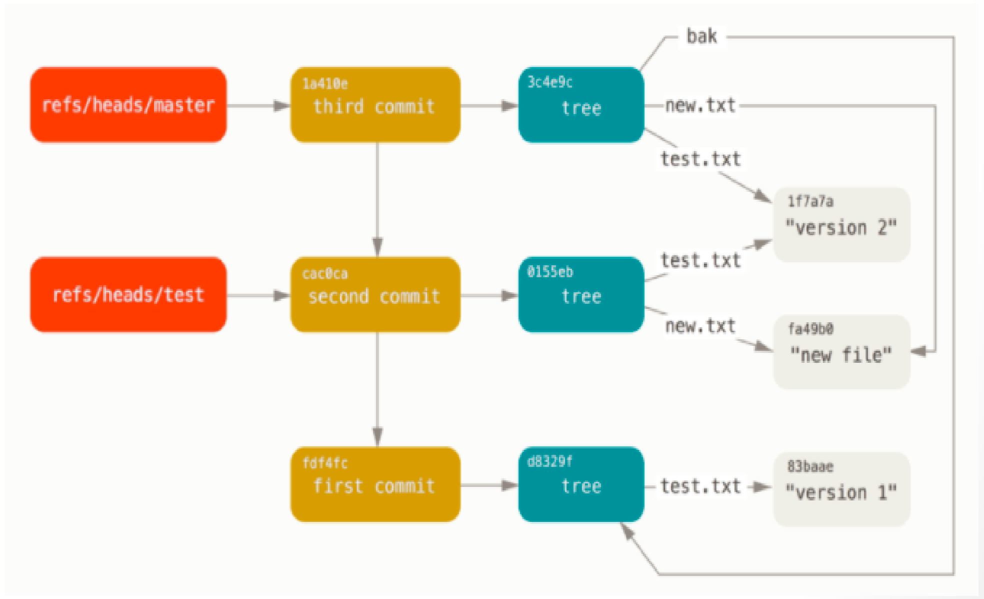


GIT Tree objects

```
$ git cat-file -p master^{tree}
```

Git stores content in a manner similar to a UNIX file system. All the content is stored as tree and blob objects, with trees corresponding to UNIX directory entries and blobs corresponding more or less to inodes or file contents





Thank You