

Version Control System using Git

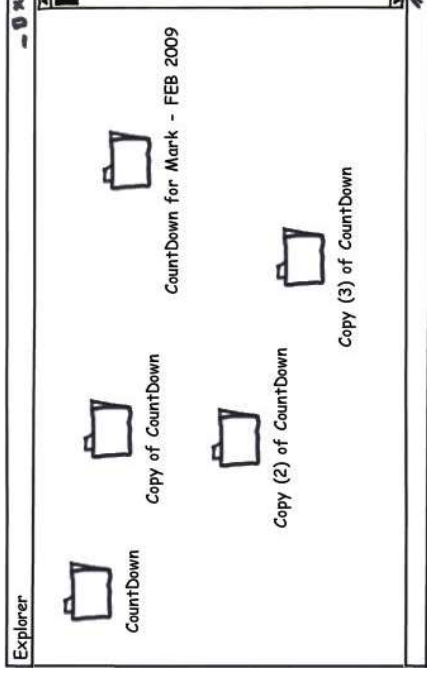


1. The Problem

Why do we need
Version Control
System anyway?

“

- Maintaining group Projects
- Patches are mostly sent via email
- Difficult to roll back
- Almost impossible to maintain if the number of people working in the project is large
- Testing new unstable features



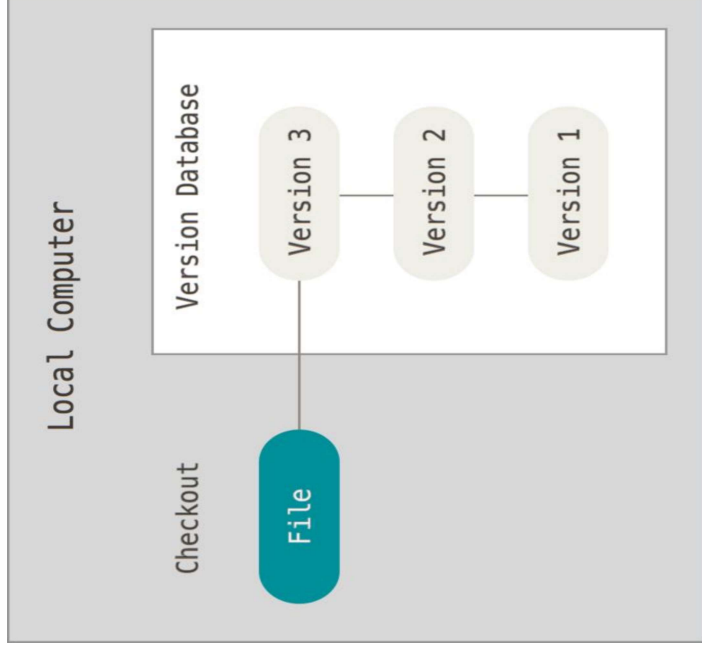
2. Version Control System



“

A Version Control System is a tool you use to track, make, and manage changes to your software code. It's also called source control.

A version control system helps developers store every change they make to a file at different stages so they and their teammates can retrieve those changes at a later time.



“

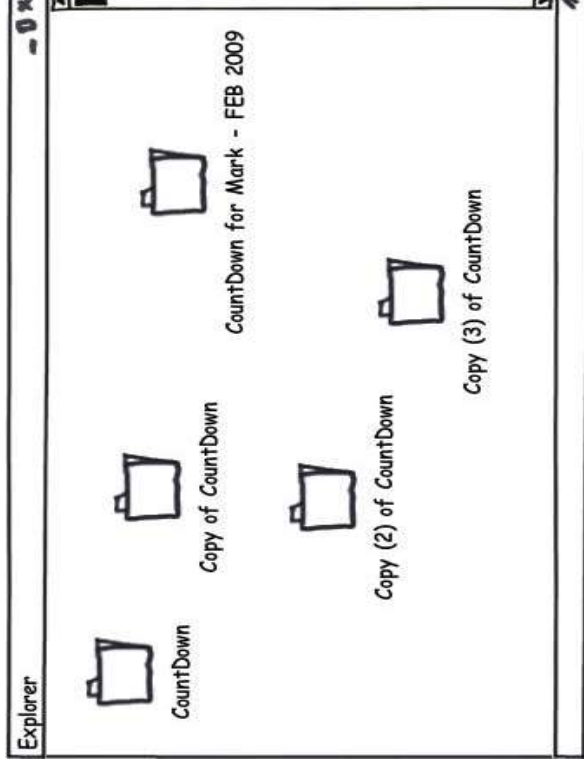
There are three types of version control systems, which are:

- Local Version Control Systems
- Centralized Version Control Systems
- Distributed Version Control Systems.

What is a Local Version Control System (LVCS)?

This is a type of version control system that is very common and simple to use. But this method is quite prone to errors and attacks because the files are being stored in your local system.

This means that you might lose the system file or accidentally forget the directory/folder of the file you are working (and then write in another directory).



What is a Centralized Version Control System (CVCS)?

In this type of version control, a server act as a repository that stores each version of the code. The CVCS helps different developers collaborate together.

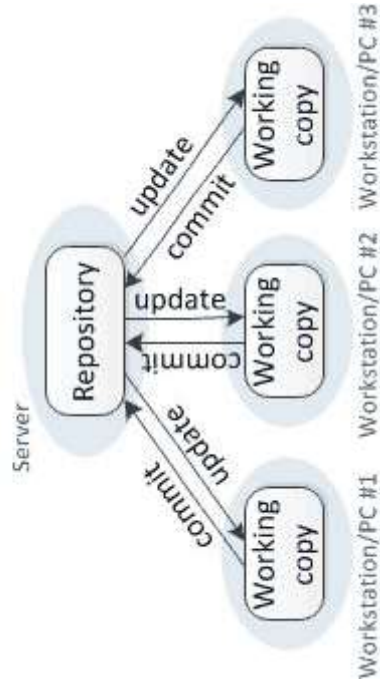
Despite the helpful collaboration and communication between developers, if a server goes down for few seconds or gets corrupted, there's the chance you'll lose your work. Unfortunately, this is a very big problem with the CVCS.

In CVCS, only a few developers can work together on a project.

The most common centralized version control systems are Concurrent Versions System (CVS), Perforce, and Subversion (SVN). There's also Microsoft Team Foundation Server (TFS), which is now known as Azure DevOps Server.

Centralised VCS

Centralized version control



What is a Distributed Version Control System (DVCS)?

This is the latest and most commonly used type of version control system these days.

In a DVCS all developers have a full back up (clone) of all the data in the server. This means that whenever the server is down or faulty, you can still work on your project and you can copy or back up your repositories to the server to restore them.

When you're using a DVCS, many developers can work together on a project. One popular DVCS is Git, which we'll talk about more now.

The three most popular of these are Mercurial, Git and Bazaar.

These systems do not necessarily rely on a central server to store all the versions of a project's files. Instead, every developer “clones” a copy of a repository and has the full history of the project on their own hard drive.



--everything-is-local

Git is a free open source distributed version control system you can use to track changes in your files. You can work on all types of projects in Git, from small to large.

With Git, you can add changes to your code and then commit them (or save them) when you're ready. This means you can also go back to changes you made before.

Git works hand in hand with GitHub



git

- Free, open source
- Fully distributed
- Handle small files very effectively
- Tracks contents, not files
- Data = Snapshot
- No network
- Three stages



git

- Created by Linus Torvalds in less than 2 weeks
- Currently maintained by Junio C Hamano



git

Git-Hub

- GitHub is a web interface where you store your Git repositories and track and manage your changes effectively. It gives access to the code to various developers working on the same project. You can make your own changes to a project at the same time as other developers are making theirs.
- If you accidentally mess up some code in your project while making changes, you can easily go back to the previous stage where the mess has not occurred yet.

Git: Stages

Three stages:

- Working directory
- Staging directory
- Git directory (repository)

Git: Stages



Git: Development

Setup

- git init
- git clone <remote-url>

Git: Development

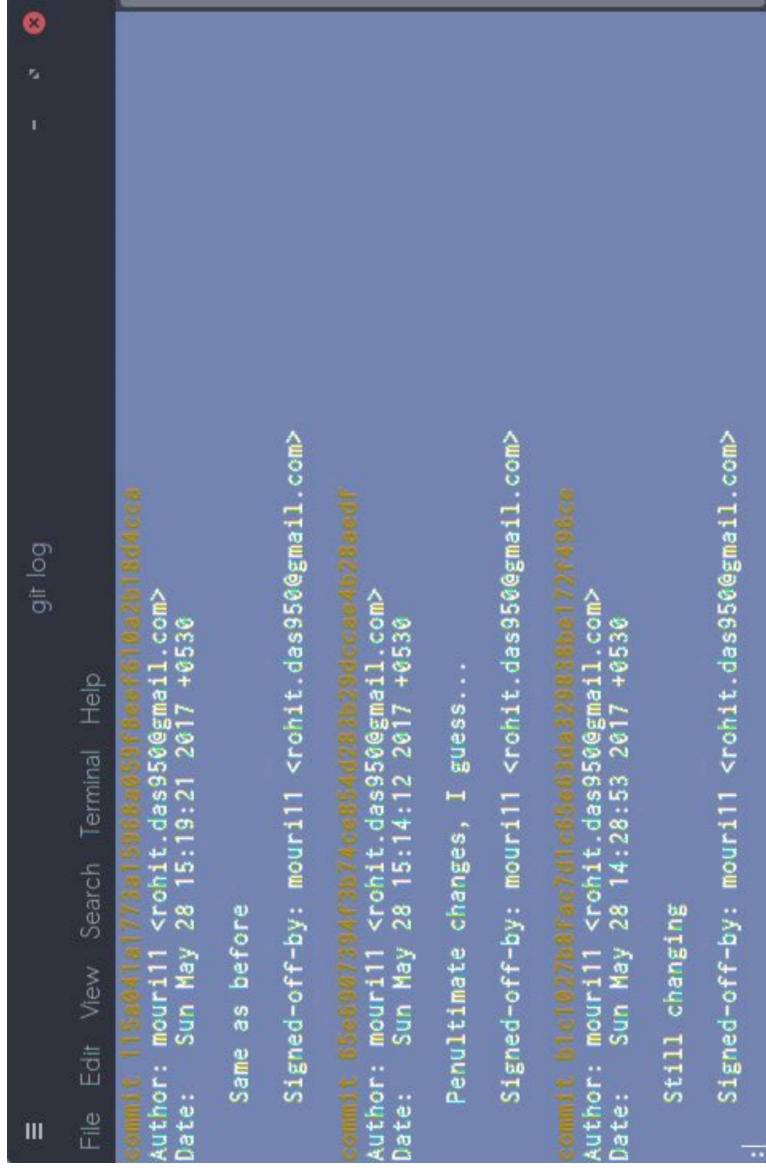
Check “snapshots” of the codebase

- git log

Show commit logs

Git: Development

Commit logs

A screenshot of a terminal window with a dark blue background and white text. The window title is "git log". The menu bar at the top includes "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal output shows a series of Git commit logs. Each log entry includes the commit hash, the author's name and email, and the date. The logs are separated by "Signed-off-by:" lines. The output ends with a colon and an exclamation mark.

```
git log
commit 115a041a1773a15908a059f8eef610a2b18d4cca
Author: mouril1 <rohit.das950@gmail.com>
Date: Sun May 28 15:19:21 2017 +0530

Same as before

Signed-off-by: mouril1 <rohit.das950@gmail.com>

commit 65e6007394f3b74ce854d283b29dcca04b28aedf
Author: mouril1 <rohit.das950@gmail.com>
Date: Sun May 28 15:14:12 2017 +0530

Penultimate changes, I guess...

Signed-off-by: mouril1 <rohit.das950@gmail.com>

commit b1c1027b8fac7d1e65e63da329038be172f496ce
Author: mouril1 <rohit.das950@gmail.com>
Date: Sun May 28 14:28:53 2017 +0530

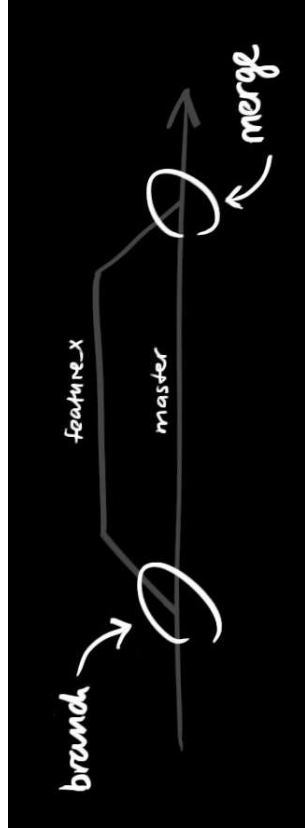
Still changing

Signed-off-by: mouril1 <rohit.das950@gmail.com>
:!
```

Git: Development

Branches

- `git checkout -b <branch-name>`



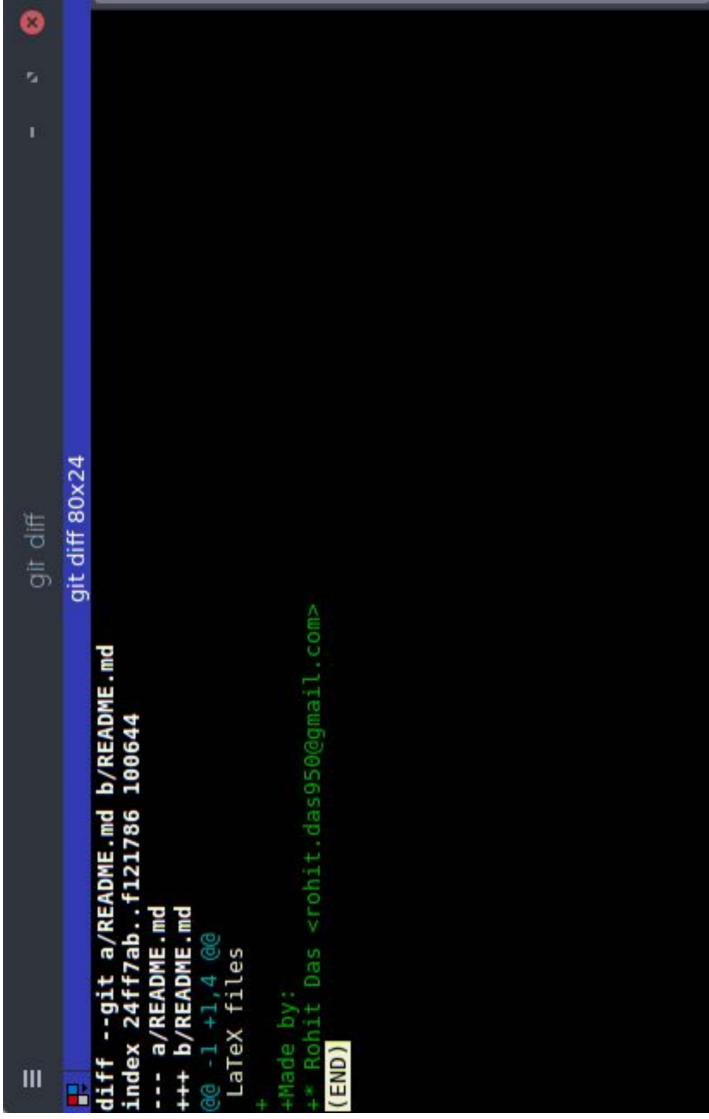
Git:
Development

View changes

- git diff

Git: Development

View changes



```
git diff
diff --git a/README.md b/README.md
index 24ff7ab..f121786 100644
--- a/README.md
+++ b/README.md
@@ -1,4 @@
 LaTeX files
+
+Made by:
+* Rohit Das <rohit.das950@gmail.com>
(END)
```

Git: Development

Update staging area

- `git add <files>`

Add file **contents** to the index

Git: Development

Create “snapshots” of your codebase

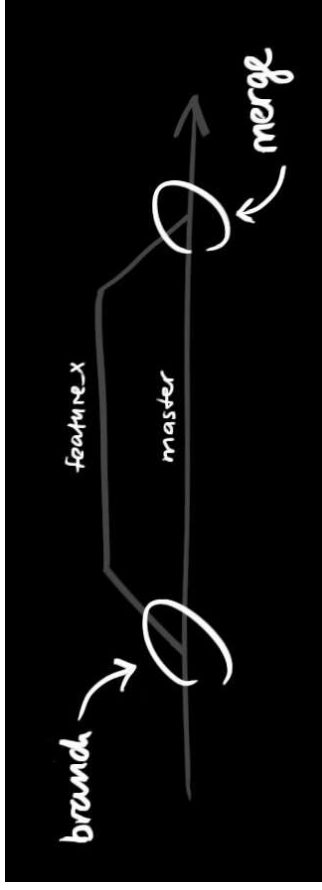
- git commit

Records changes to the repository

Git: Development

Merge other branches

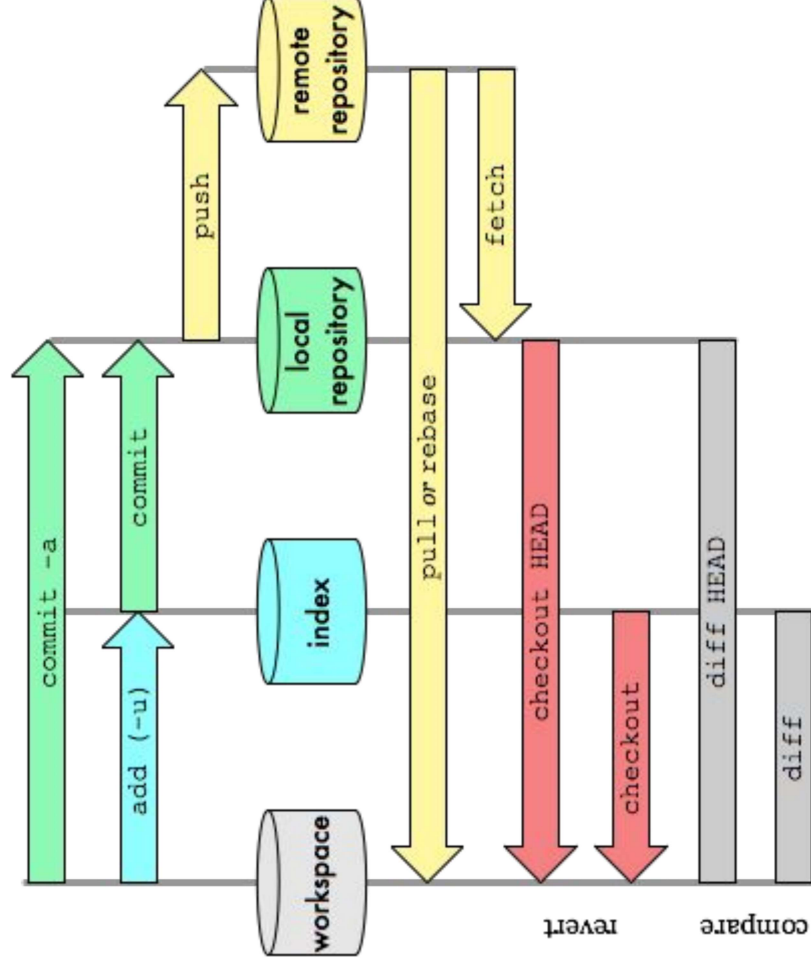
- git merge



Git: Development

Git Data Transport Commands

<http://osteele.com>



Git: Development

Make patches

- `git format-patch --stdout > fix.patch`

Patch created as "fix.patch"

Prepare patches for email submission

Send patch via mail

Git: Development

Make patches

```
git format-patch master --stdout
git format-patch master --stdout 80x24
From 027c42c2436f5c06077619e2338d82d2baacd526 Mon Sep 17 00:00:00 2001
From: mourill <rohit.das950@gmail.com>
Date: Mon, 29 May 2017 11:17:58 +0530
Subject: [PATCH] readme

Signed-off-by: mourill <rohit.das950@gmail.com>
---
 README.md | 3 +++
 1 file changed, 3 insertions(+)

diff --git a/README.md b/README.md
index 24ff7ab..f121786 100644
+++ b/README.md
@@ -1,4 @@
 LaTeX files
+
+Made by:
+* Rohit Das <rohit.das950@gmail.com>
--
 2.7.4
(END)
```

Git: Development

Applying patches

- `git apply < fix.patch`

Applies changes from the patch

Result?

- Much efficient workflow
- Creating and merging branches are very easy and fast

Result?

The development process of the Linux kernel is maintained using Git

The Linux kernel development process has:

- Over 2000 individual contributors per year
- Grows by nearly 300,000 lines per year

Result?

