Introduction to Git

Naoki Pross — np@0hm.ch

XX. March 2025

Table of Contents

1. The Problem
2. The Solution
3. The Implementation
4. Using Git

What do we want?

The Problem

Synchronize data across multiple computers, with multiple people working on (possibly the same) files.

Linus' Wishes (The guy who invented Git)

- Synchronization *always* works
- Teamwork is possible and efficient
- Works offline
- Fast

intuitive or easy to use was not on his list!

Other Solutions?

Popular at Linus' Time

CVS Slow to synchronize. CVS requires a centralized server which can get overloaded, was usually set up by the company IT.

E-Mail People sent patch files to each other via email.

Popular Tools Today

Cloud Storage Does not work offline. Their whole business model is against you. You have no (real) control over when to sync. Also, sharepoint is garbage.

Mercurial (hg) Learn to walk (Git) before you run.

Table of Contents

1 The Problem

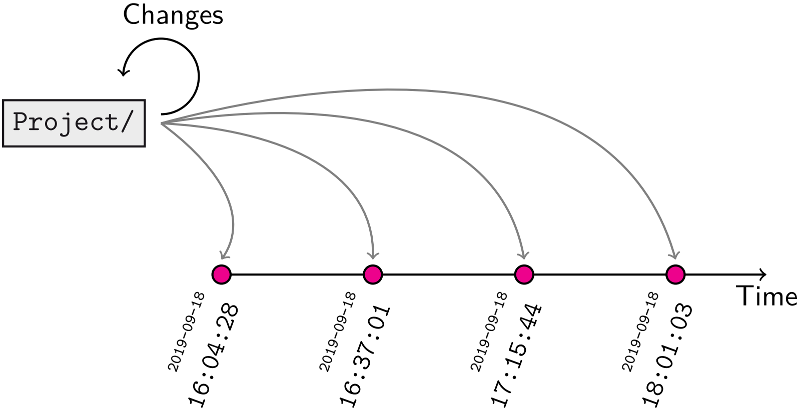
2 The Solution

- Concurrent Changes: Commit Graph
- More Files: Blobs and Trees
- Concurrent Changes: Branches and Merge
- Many Computers: Remotes

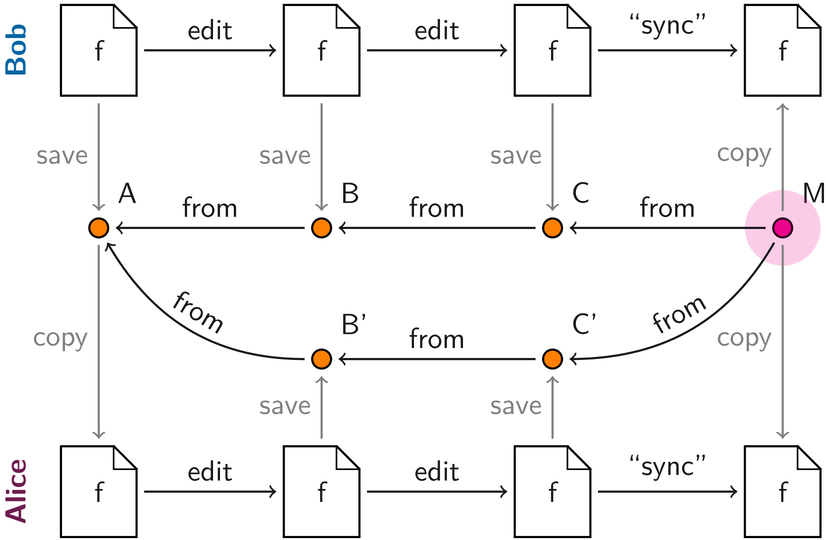
3 The Implementation

4 Using Git

Solving the Problem: Snapshots



Solving the Problem: Concurrent Changes I



Solving the Problem: Concurrent Changes II

High Level Overview

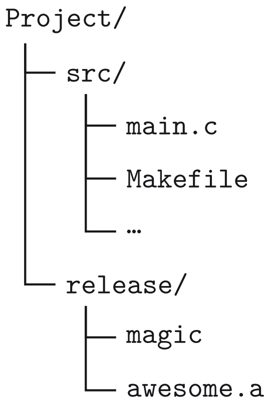
Store changes using a *directed acyclic graph* (DAG) called the *commit graph*.

- Nodes are saved points in time called *commits*
- Arcs point to state from which change was made
- Commits with multiple children (A) are *branching commits*
- Commits with multiple parents (M) are *merge commits*

Problems

1. We care about file content not the files itself
2. Alice and Bob are not working on the same computer
3. How do we merge changes?

Solving The Problem: Multiple Files



Project/

src/

main.c

Makefile

…

release/

magic

awesome.a

Filesystem Jargon

Tree Folder / Directory

Blob Binary Large OBject, raw data (bits) of file content[a]

File Blob + Metadata (Name, Date, …)

Solution

Treat blobs as single entity with metadata. Examples:

- Rename file ⇒ Same blob, commit name change
- Move file ⇒ Same blob, commit change tree

[a] Demo: `hexdump` vs `stat`

Mathematical Digression: DAG

Directed Acyclic Graph

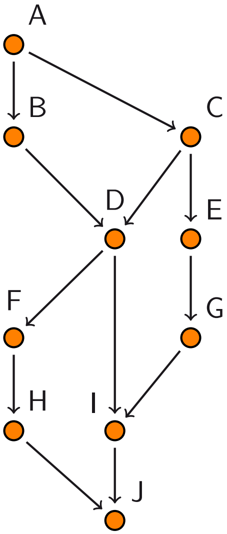
A DAG $G = (V, A)$ is defined by a finite set of vertices V and a finite set of *arcs* A and may not contain loops.

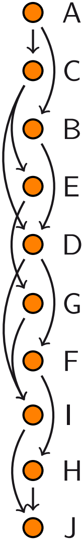
Partial Order

DAG have a partial order relation $u \succ v$ for comparable $u, v \in V$.

Topological Order

A DAG $G = (V, A)$ has a total order $\succ^*$ by having that for all $(u, v) \in A$ $u \succ^* v$. If G has a Hamiltonian path $\succ^*$ is unique.





Solving The Problem: Concurrent Changes III

Solving The Problem: Concurrent Changes IV

Solving The Problem: Other Computers I

remote branches

Solving the Problem: Other Computers II

push and fetch and pull

Table of Contents

1 The Problem

2 The Solution

3 The Implementation

- Hash and Merkle DAG
- The Interface of Git

4 Using Git

Mathematical Digression: Hashes and Merkle DAG

“One-way fast” functions

Hash Function

A (cryptographic) *hash* function is an $h : \Omega \to \{0,1\}^d$ for a fixed hash length d such that:

1. Given $y = h(x)$ it is hard to find x
2. It is hard to find $x, y \in \Omega$ s.t. $h(x) = h(y)$
3. Given $h(x)$ it is hard to find y s.t. $h(x) = h(y)$
4. Given $h(x)$ and a function f it is hard to find $h(f(x))$

Hashes are *not* unique!

Merkle DAG

A Merkle DAG is a DAG $G = (V, A)$ with a hash

$$h : U \subseteq V \to \{0,1\}^d$$

that defines a label function

$$\ell(v) = h(\{v\} \cup \mathrm{neighbors}^+(v))$$

Properties

- Immutable data structure
- Has cryptographic verification

Mathematical Digression: Visualizing Merkle DAGs

Git Commits

Commit Contents

- Content (Blobs and Trees) hash
- Parent(s) commit(s) hash(es)
- Metadata: Author, Date, Message

Example

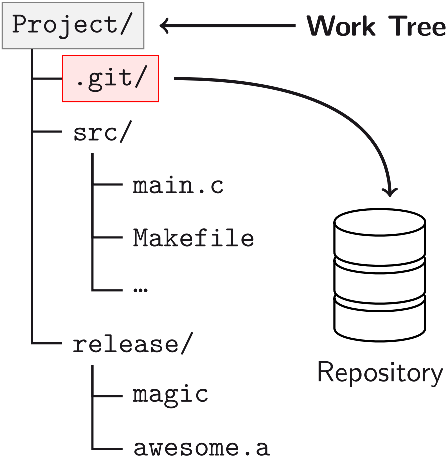
```
commit 1cfdf5c198f1c74c2f894067baf4670f5bca8e70
Author: Nao Pross <np@0hm.ch>
Date:   Wed Feb 9 19:53:06 2022 +0100

  Fix arrayobject.h path on Debian based distros

  On Debian Linux and its derivatives such as Ubuntu and LinuxMint, Python
  packages installed through the package manager are kept in a different
  non-standard directory called 'dist-packages' instead of the normal
  'site-packages' [1].

  To detect the Linux distribution the 'platform' library (part of the
  Python stdlib) provides a function 'platform.freedesktop_os_release()'
```

Git Repositories



Project/

Work Tree

.git/

src/

main.c

Makefile

…

release/

magic

awesome.a

Repository

Work Tree

Root of your project, contains `.git`. **Never delete `.git`.**

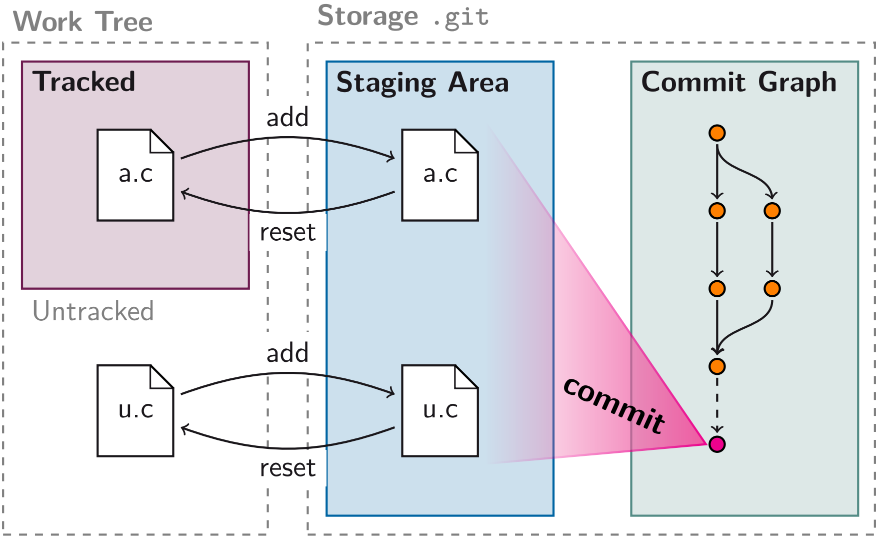
Repository

- Commit graph (Blobs, …)
- Staging Area (will come next)

Table of Contents

1. The Problem
2. The Solution
3. The Implementation
4. **Using Git**
   - The Conceptual Areas
   - Branches and Merges
   - Best Practices
   - Forks
   - GitHub and Others

The 3 (or 4) Conceptual Areas of Git



Branches, Remotes and your HEAD

What is a Commit?

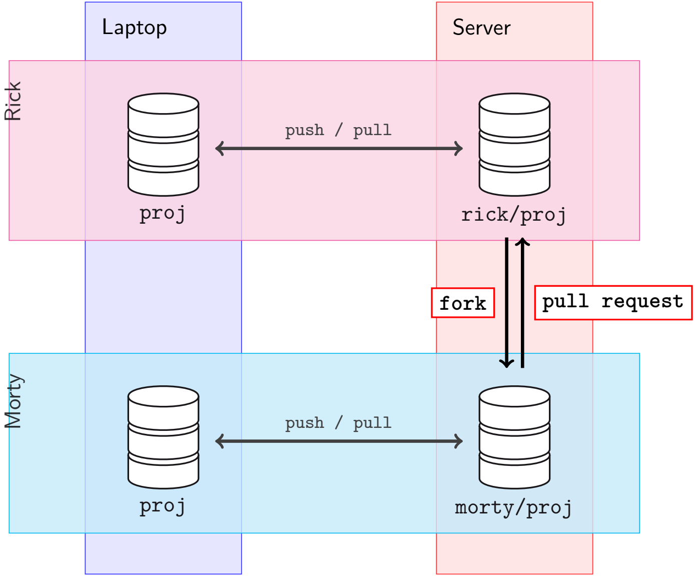
Trunk, Feature Branches

Releases and Tags

Forking Projects

Git Services (GitHub, GitLab, ...)

Forking and Pull / Merge Requests



Laptop
Server
Rick
proj
push / pull
rick/proj
fork
pull request
Morty
proj
push / pull
morty/proj