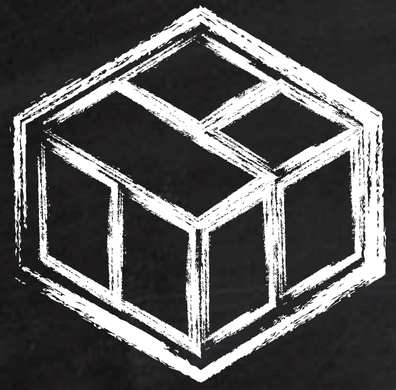


Architecting and Engineering Main Memory Database Systems in Modern C

Chapter 1
A Full-Stack Engineer's Basic Kit

Marcus Pinnecke, M.Sc.
Research associate / Working Group Database & Software Engineering
Institute of Technical and Business Information Systems (ITI)

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

The most projects I started,
started with the one thought…

„Yeah, can do. Shouldn't be that hard."

… and then you end up with writing an entire database system ;)

Hofstadter's law

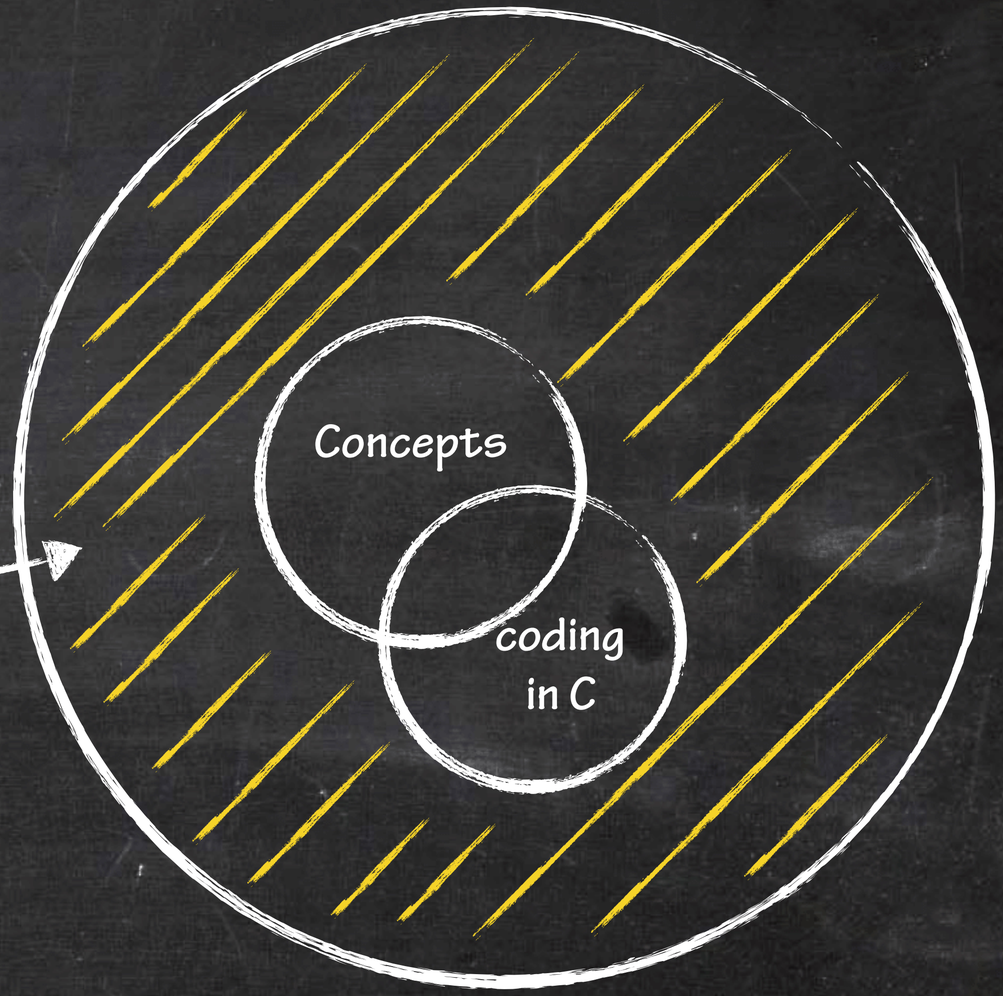
> “It always takes longer than you expect, even when you take into account Hofstadter's Law”
>
> — Douglas Hofstadter (1979)

If a task is somewhat complex, it’s highly probable that you will not estimate the effort correctly.

This includes the effort to understand that the task is somewhat complex

What's this Chapter about?

Stuff and Skills that „surrounds" the act of designing, engineering and coding



nmentation

programming paradigms

architectural patterns

build server

infrastructure & devops

debugging

team communication

design patterns

dependency management

build systems

version control

project management

profiling

ticketing systems

best practices for coding

maintenance

licensing

testing

modeling w/ UML

reverse engineering

collaboration

refactoring

tuning

documentati

DISCLAIMER

The following content is intended as an overview on topics that you'll face during work on software projects.

This lecture provides you the basics in order to get you productive (not to make you an expert).

For more details and more depth, visit dedicated lectures and consider to read other material.

VERSION CONTROL

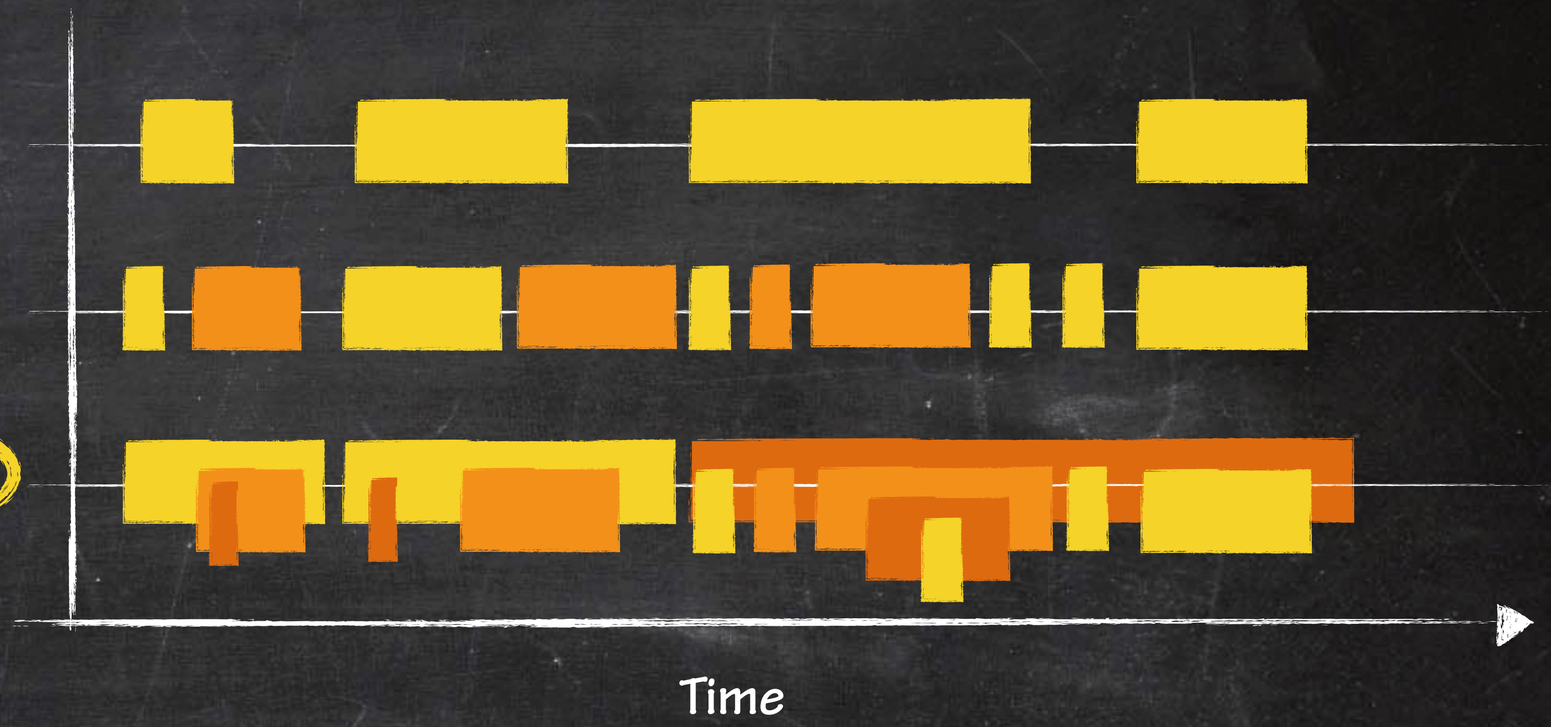
Motivation

only you — asset base

exclusive you and others — asset base

non-exclusive you and others — asset base

That's professional and effective software development



changes made by you

changes by others

Motivation

guy 1 guy 2 guy 3 guy 2 guy 5 guy 1

1

changes to the asset base

(FINAL) PROJECT ASSET BASE

2

changes to the asset base

guy 5 guy 1 guy 3 guy 1 guy 2

How to organize, sync, track and back-up code in a common repository?

What are Version Control Systems (VCS)?

- Developer tools to manage (team-based) working on a (software) project, they help to:
  - Organize teams of developers writing and changing code continuously
    - Team mates should not block each other
  - Track history of any change over time (commits)
    - Transparent overview about changes made, contradictions,...
  - Store everything in a kind of a database (repository)
    - Also saves space by compression of history (changes,...)
  - Enables to compare different versions and commits, track changes, sources of bugs, undo changes, apply changes from others,...
  - Conflict-free* concurrent work on shared resources (e.g., source code)

*almost

Version Control & Version Control Systems (VCS)

save change to file

work on changed file

history and content of all versions

repository

called commit

called checkout

diff

file difference to certain version

working copy

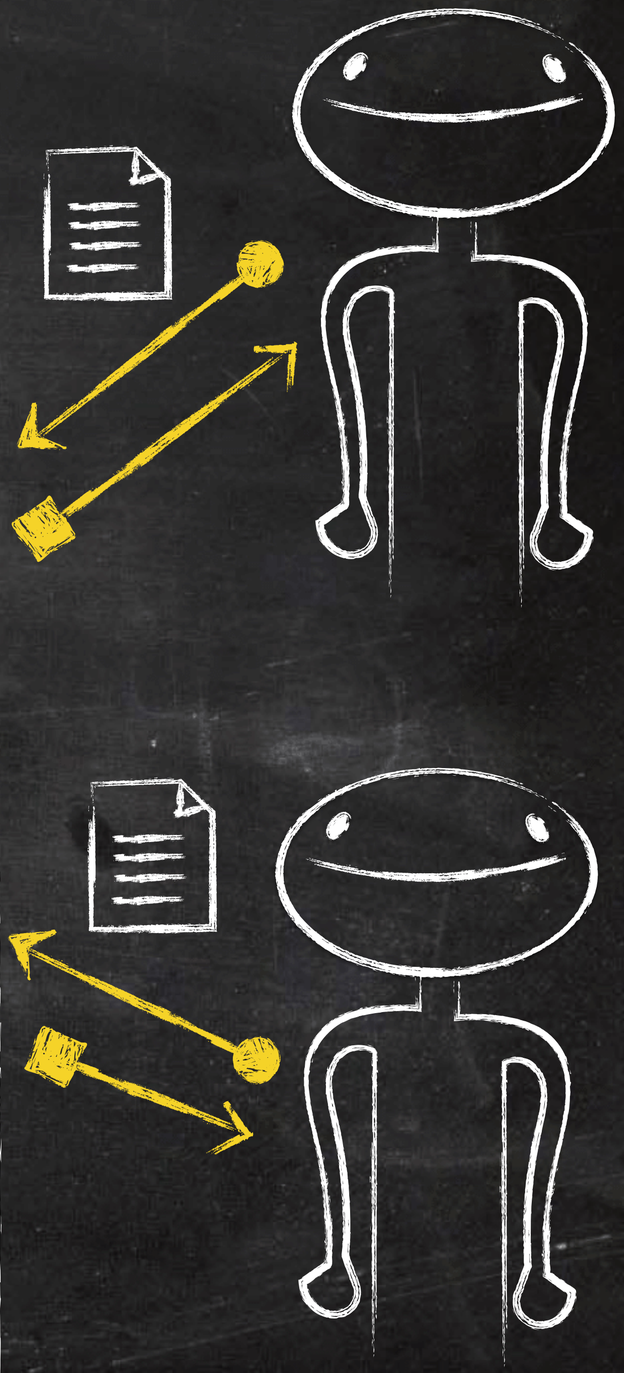
copy of history and content of a certain version

VCS

Asset Base

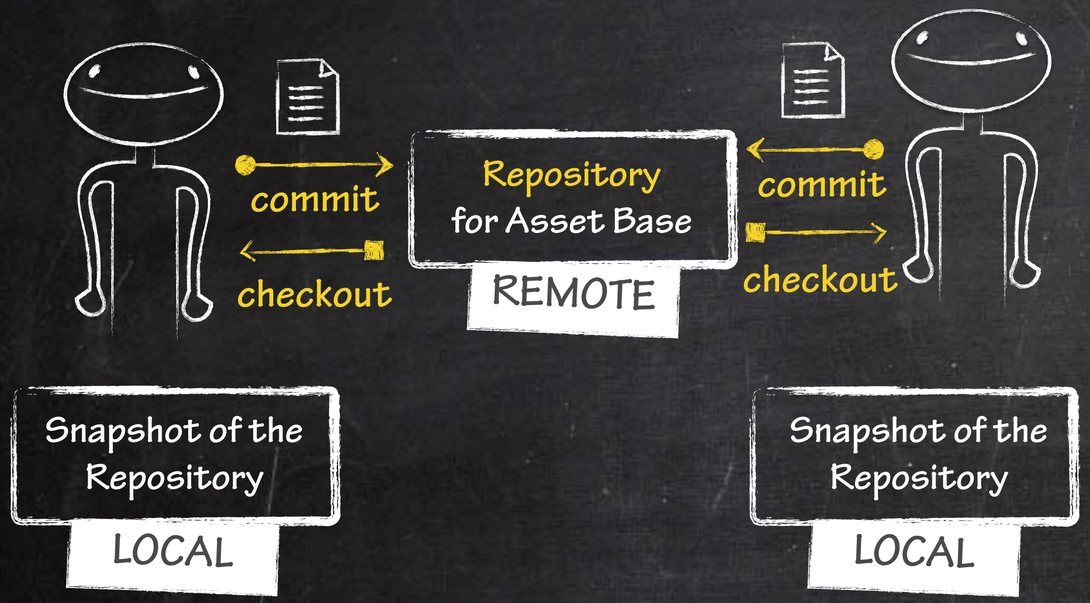
Code, Files, Images,...

- Collaboration: organizes multiple revisions (versions) on the (same) shared asset by multiple users in collaboration
- Workflow: defines protocol on how changes are made by one, and fetched by others, and where the asset base is located
- History: tracks history of changes and versions
- Recovery and Sharing: allows fine-grained undoing of changes and redoing changes between versions

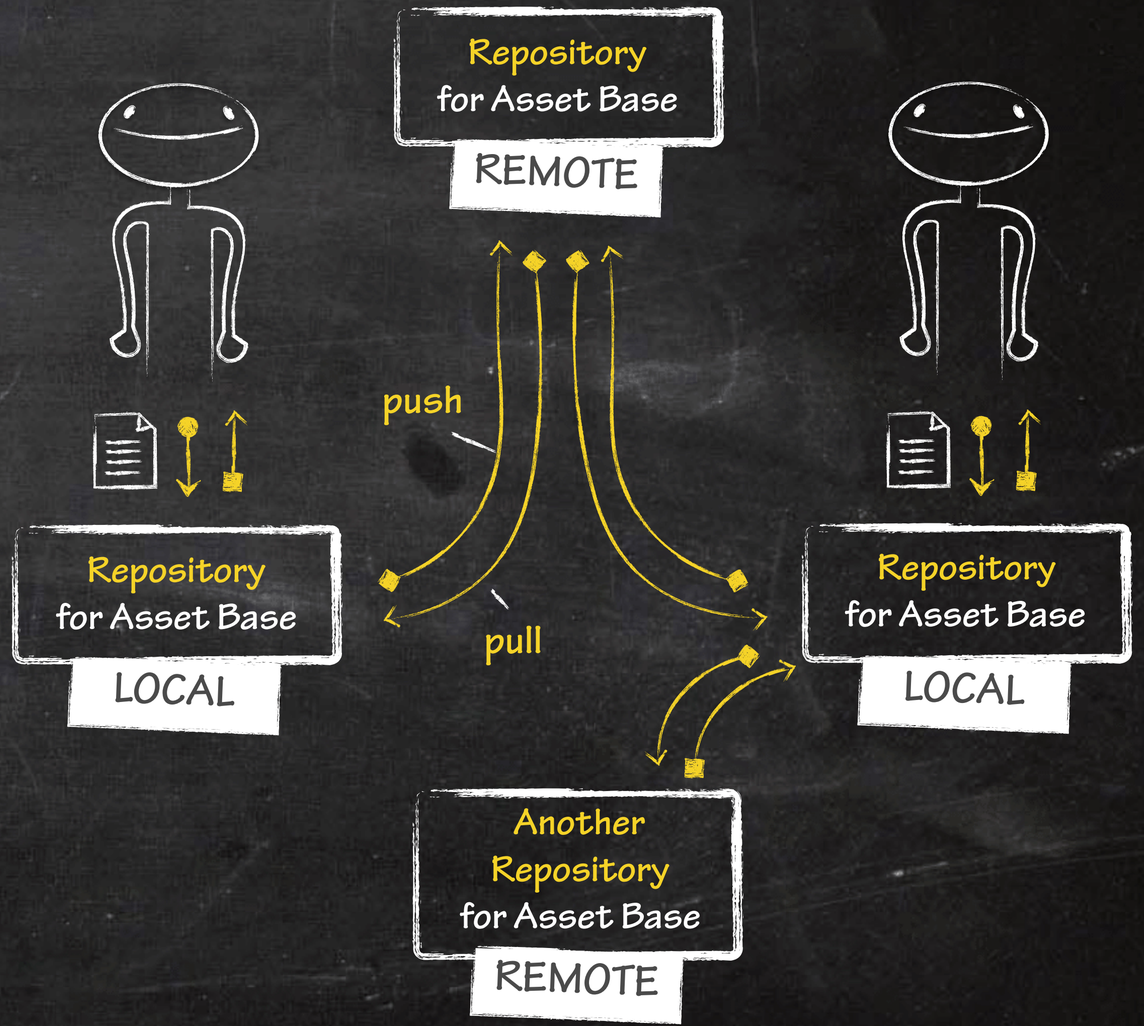


VCS Types

CENTRALIZED



DECENTRALIZED



Which VCS Type to chose?

VCS it's a kind of communication protocol…

Depends on the kind of problem to solve!

… and often decided at management level!

CENTRALIZED

Philosophy

„There is only one source that is the product“

Contributors are expected to communicate safely and to have secure connections to the central repository!

DECENTRALIZED

Philosophy

„All repositories are equal. Which one contains the product is not in responsibility of the VCS.“

Contributors across multiple countries and time zones with no secure connections to a central repository!

VCS Types Systems

CENTRALIZED

- CVS (... it's dying)
- Subversion (... it's dying)
- Microsoft VisualSourceSafe (... not even Microsoft anymore)

- ... and more (of course!)

projects moved or moving to

DECENTRALIZED

- Git (e.g., used by Linux Kernel devs)
- Mercurial (e.g., used by Mozilla devs)
- Bazaar (e.g., used by Ubuntu dev)

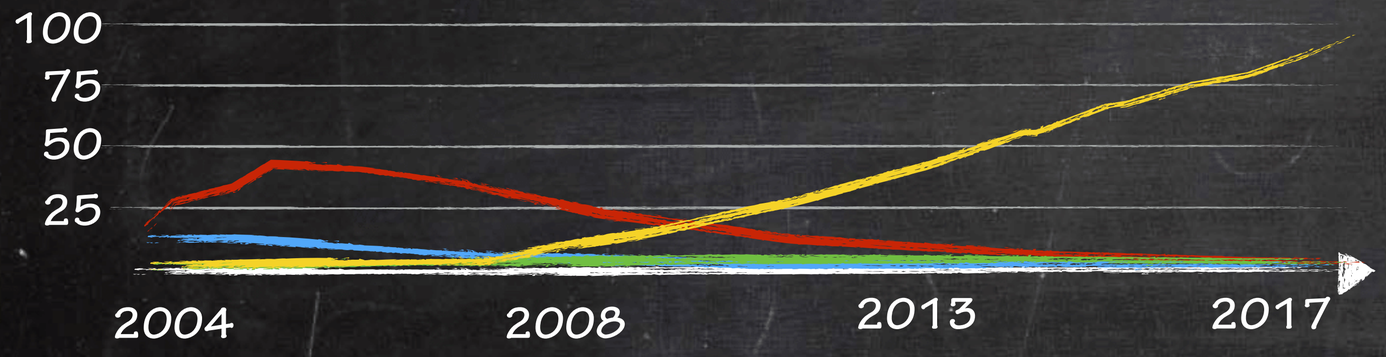
- ... and more (of course!)

Centralized VCSs are dying...

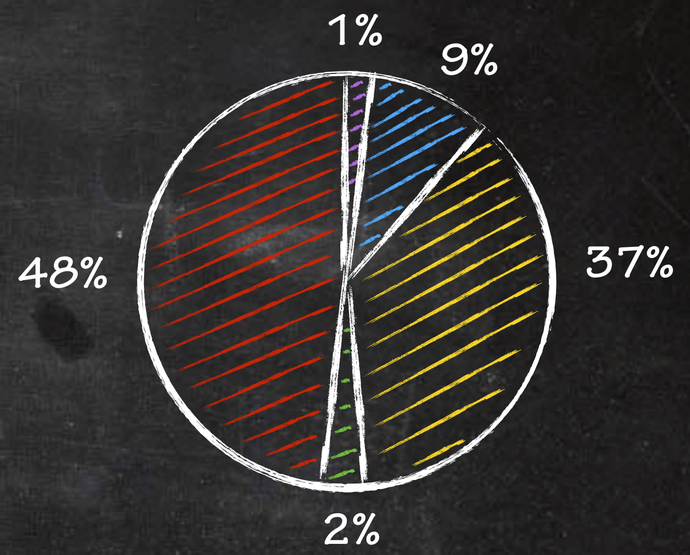
VCS Trends

Git Bazaar Subversion CVS Mercurial

Relative number of worldwide web search from 2004 to 2017 in category „developer tools“ according Google Trends (simplified)
As of October 4, 2017



Source Code Repositories from Open Hub
As of October 27, 2014



Git

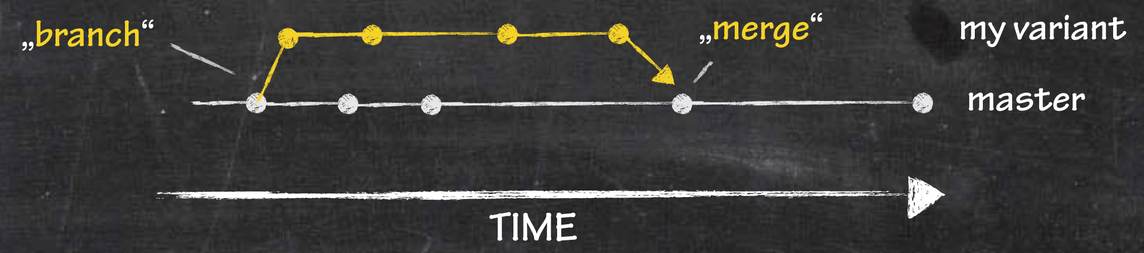
What is Git

- Free software distributed VCS initiated by Linus Torvalds
- Git is a command-line tool that is almost a „file system“ with VCS features
- Lightweight branching and merging in local/remote repositories
  - Built-in fail-safe mechanism for code integration
  - Change sharing between repositories
- Git is used to version control Linux Kernel
    - Motivation for Git was changed in the license terms of a former tool used by the Kernel community
- Fun fact: The term „Git“ is British english slang for „moron“
    - Joke by Linus Torvalds he names all his projects after himself

Git Features

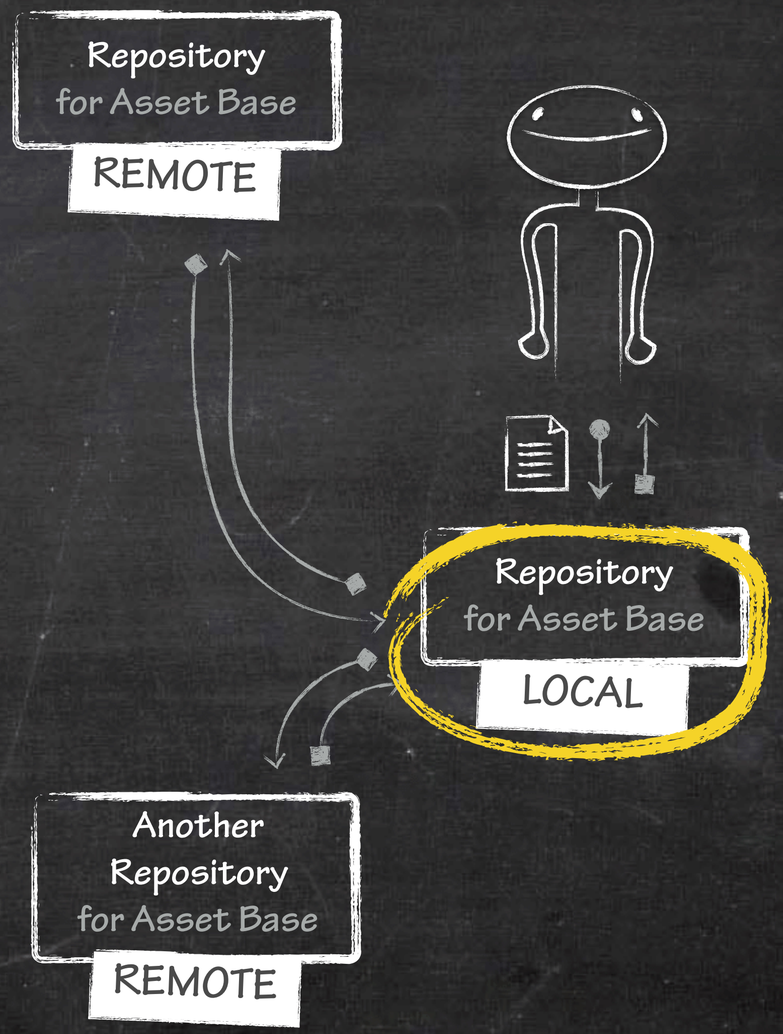
- History: Change history of every file for every commit (e.g., allows reverting changes) incl. meta data (e.g., the contributor)

- Time-travel between different version and change between different variants



- Branching and Merging: support of concurrent changes by „per-developer's“ individual stream of change that create a variant which is independent of the rest
- Tracking: each commit has a unique id that can be referred two (e.g., bug tracking)

Local Git Repositories



Local Git Repositories (2)

- A local Git repository is a dedicated directory incl. its history (.git database) on the local machine whose content can be changed
  - Marking a local directory as repository: initializing `$ git init`
  - Removing a repository: removing directory `$ rm -rf <directory>` (or the .git database)
  - Modifications on the repository elements = modification on the working tree
    add files with `$ git add` and remove files with `$ git rm`
    - Adding a collection of changes in working tree to the history: commit `$ git commit`
    - Working tree file states: `$ git status`
      - untracked files are *not* managed by git
      - tracked files are managed by git (i.e., they have a history)
      - modified files are changed but the change is not staged
      - staged files (resp. modifications) for one of the next commits

alternatively create a „patch“ file that contains your changes without performing a commit.

Local Git Repositories (3)

Versioning

- A directory (in the file system) that is marked as a git repository and managed with Git
- Git philosophy on changes to that repository: a set of snapshots (inside a branch)
  - Every commit („save“) is a picture of the current repository modification state in the git database