GIT TUTORIAL

Creative Software Architectures for Collaborative Projects
CS 130

Donald J. Patterson

SOFTWARE CONFIGURATION MANAGEMENT
SOURCE CODE MANAGEMENT

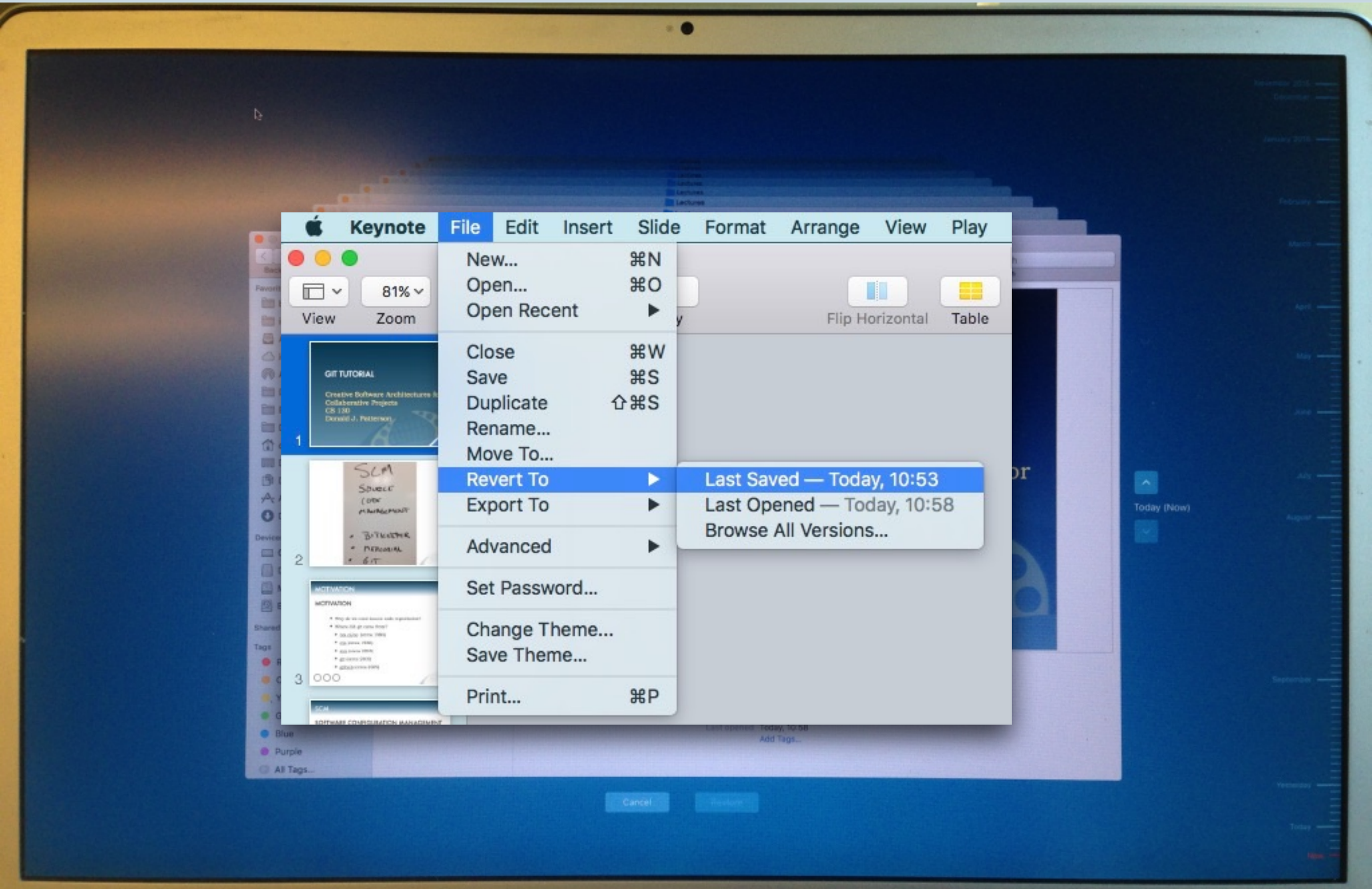
- Generic term for the ability to manage multiple versions of
  - a document
  - a collection of documents

SOFTWARE CONFIGURATION MANAGEMENT

"BUILT-IN" EXAMPLES

- Microsoft Word
  - backup copies or "auto-save"
- MAC OS X
  - Time Machine
  - Revision Management
- Wikipedia
  - Page History
- Google Docs
  - Change History

SOFTWARE CONFIGURATION MANAGEMENT



MOTIVATION

GENERAL

- Backups
  - Restore against disaster
- Robust backups
  - Restore against vandalism
- Backups for collections
  - Recognizing the dependencies between files
  - Preferences and configuration files also

MOTIVATION

SOFTWARE AS A SPECIFIC CASE

- A “version” of software refers to a collection of files
  - source code
  - graphics assets
  - project build configuration files
  - libraries
- These have to all be in sync or the project won’t build and run

MOTIVATION

SOFTWARE AS A SPECIFIC CASE

- At the same time multiple versions are in the wild
  - A version that has been released
  - A version that is being worked on by developers for the next release
- A customized version for a specific client
- A version that was “forked” to go in a different conceptual direction

MULTIPLE VERSIONS OF SOFTWARE

- One could just keep a complete copy of all files for each release
  - However this means many nearly-identical copies are being kept
  - Requires a lot of discipline on the part of an organization to archive and organize
  - Careful permissions need to be managed for access to read versus write versus copy
  - In some cases intellectual property also needs to be tracked

SOFTWARE CONFIGURATION MANAGEMENT

STAND-ALONE EXAMPLES

- Version Control Systems
  - Support check-in and change management of files in general
  - Automate the management of the software
- Examples
  - subversion (svn)
  - Bit Keeper
  - Mercurial
  - Git
  - Git Hub

SEMANTIC VERSIONING

AVOIDS "DEPENDENCY HELL"

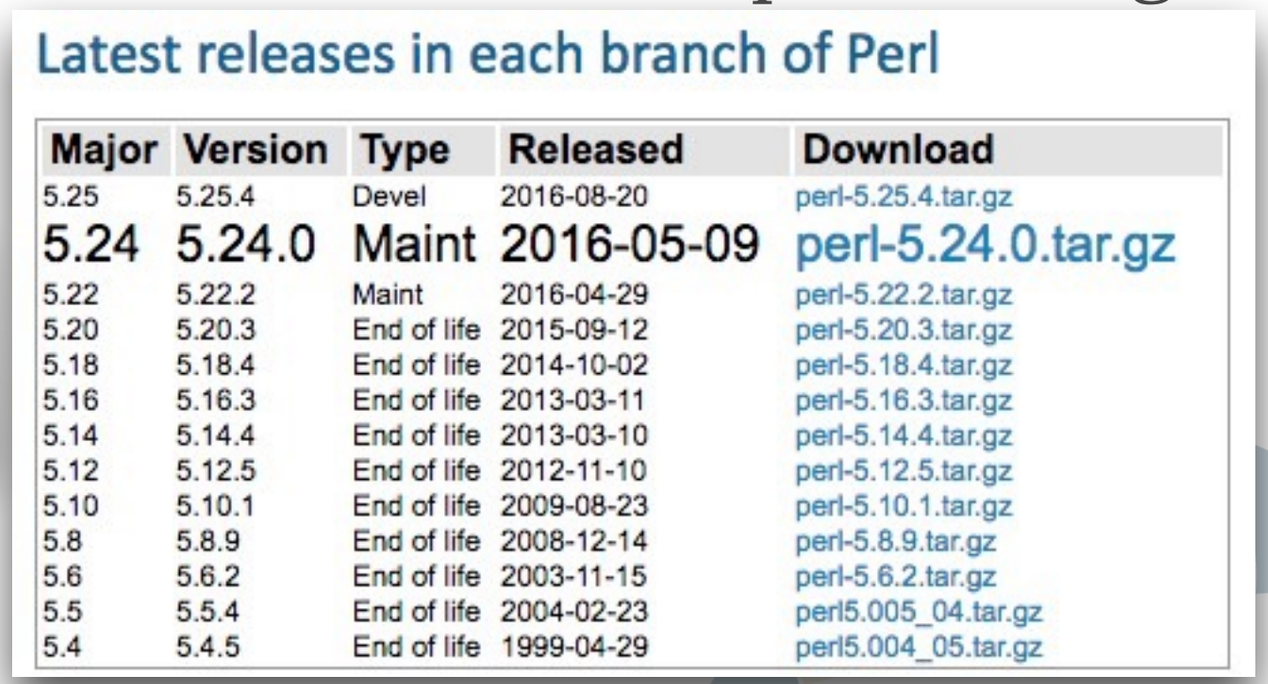
- Libraries save enormous amounts of space
- When you want to release a new version of a piece of software that depends on other software that is also constantly releasing new versions.
- Without a consistent way of understanding how versions change you might have to keep a separate version of every library for every thing that depends on it

http://semver.org/

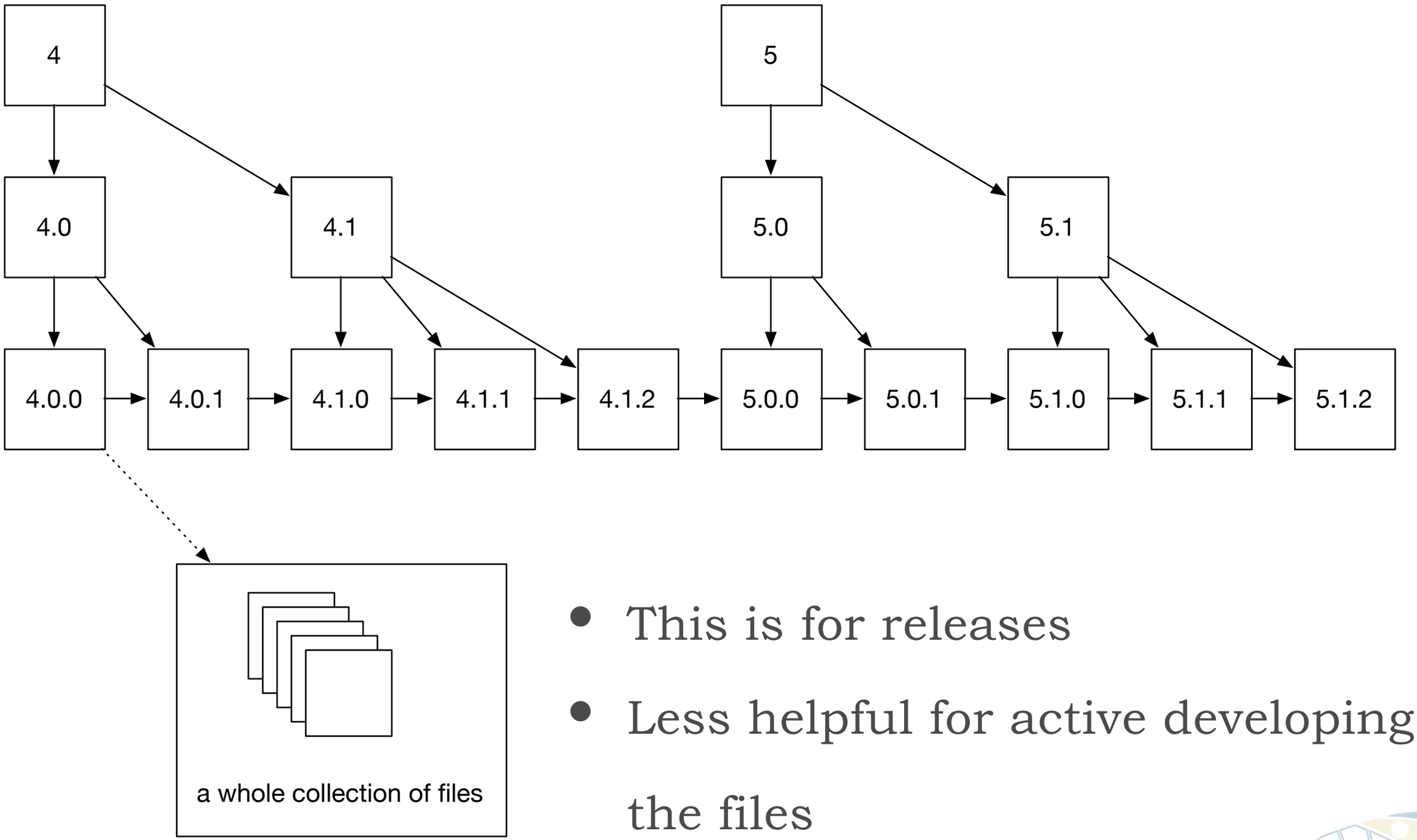
SEMANTIC VERSIONING

A NUMBERING SCHEME FOR LINEAR CHANGES

- Given a version number MAJOR.MINOR.PATCH, increment the:
  - MAJOR version when you make incompatible API changes,
  - MINOR version when you add functionality in a backwards-compatible manner, and
- PATCH version when you make backwards-compatible bug fixes.



MAJOR.MINOR.PATCH



- This is for releases
- Less helpful for active developing of the files

TERMS

- Repository
  - A location where files and their history are kept
- Checking Out
  - Obtaining a copy of the file(s) in the repository
- Working copy
  - The copy of the files that you have checked out
- Checking in or “Committing”
  - Returning changed files to a repository

TERMS

- Branching
  - When files have a common parent but are being changed in parallel
- Merging
  - Combining branches into one common descendent
- Conflict
  - When merging can’t be done automatically
- Resolve
  - Fix conflicts

Time

Trunk

1

Branch

2

3

Merge

4

Release

Branch

T1

5

6

7

Merge

9

8

Release

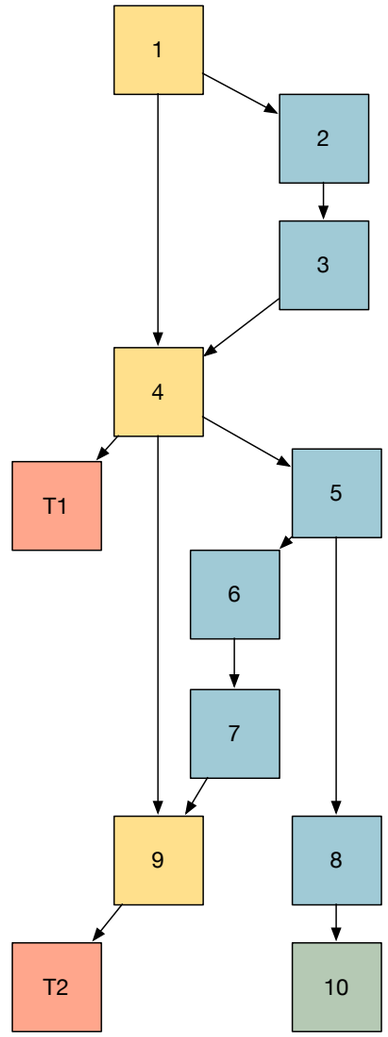
T2

10

Fork

REPRESENTATION

- Over time changes to a source code repository
  - are not a “linked list” because of branches
  - are not a “tree” because of merges
  - are not a “directed graph” because time prevents cycles
- are a “directed acyclical graph” or DAG



TERMS

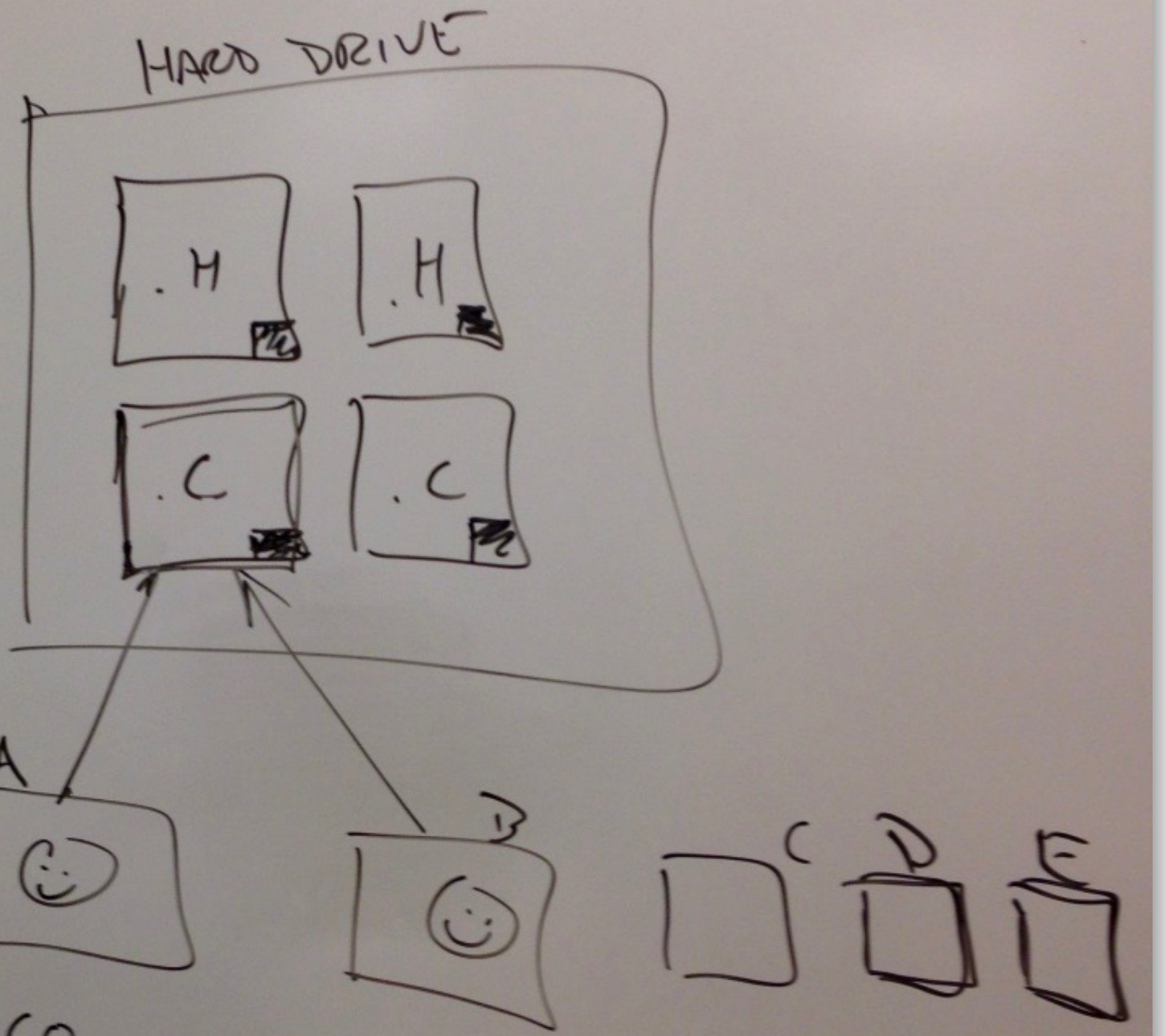
- Clone
  - Copying an entire repository, history and all
- Delta compression
  - Only keeping information about changes between each branch, commit or merge
- Tag
  - To label a commit for future reference
- Push and Pull
  - Sync repositories with each other

HISTORICAL PERSPECTIVE

- How did we get to where we are today?
  - rcs ci/co (cerca 1980)
  - cvs (cerca 1986)
  - svn (cerca 2004)
  - git (cerca 2005)
  - github (cerca 2009)

RCS

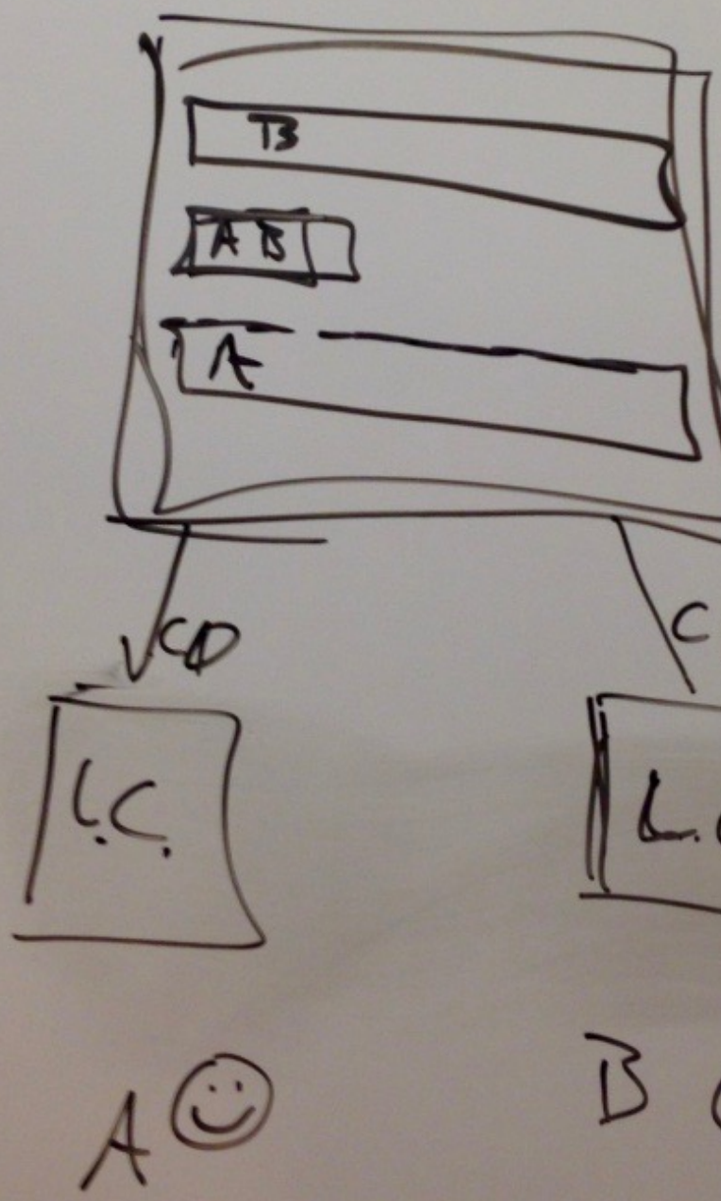
- FILE LEVEL LOCKING
- SHARED FILESYSTEM



CVS

- SMART MERGING
- BRANCHING

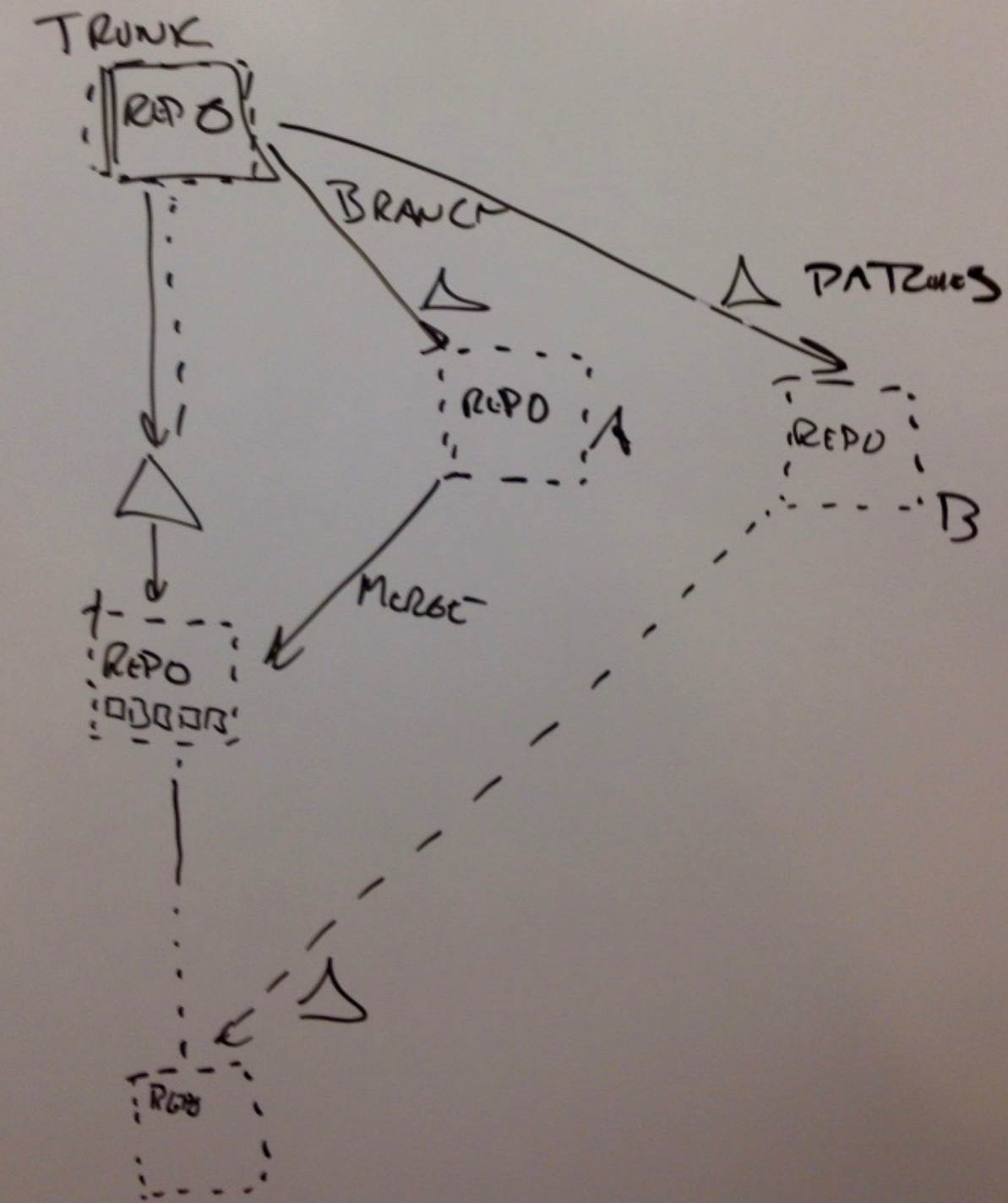
DIFF
PATCH

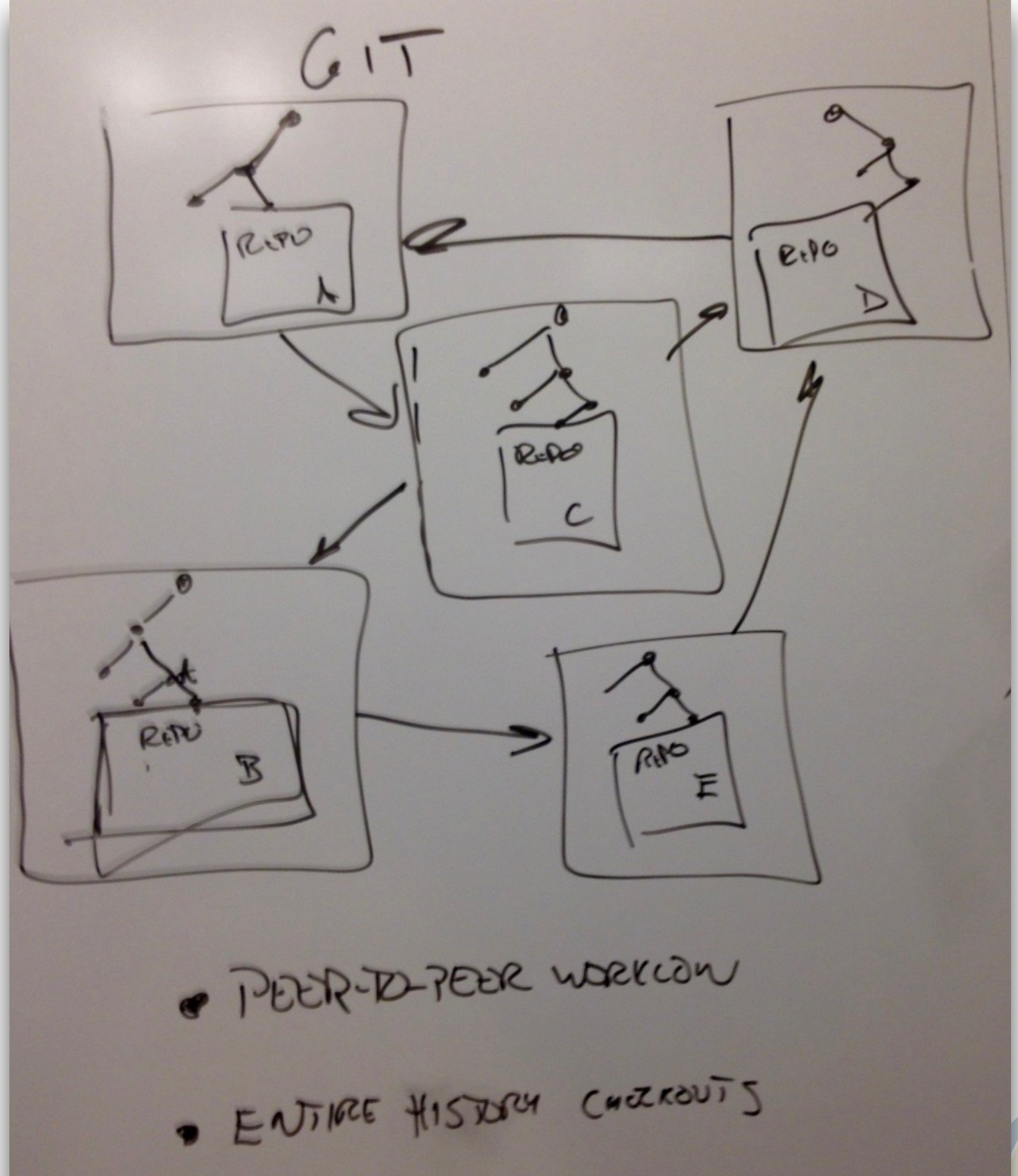


"SUBVERSION"

SVN

- TRUNK (WORKFLOW)
- SMART BRANCHING




GIT
REPO
A
REPO
B
REPO
C
REPO
D
REPO
E
• PEER-TO-PEER WORKFLOW
• ENTIRE HISTORY CHECKOUTS

GITHUB

- WEB HOSTED REPOSITORY STORAGE
- COLLABORATION SITE
- FORUMS DISCUSSING CHANGES IN REPO INITIATIVES
- CLOUD
- QUALITY / SECURITY SYSTEM

TOOLS

GUI FOR GIT

- https://www.atlassian.com/software/sourcetree

ACCOUNT ON GITHUB

- https://github.com/

CODE IN THE WILD

- Let’s look at a project in the wild
  - bitcoin

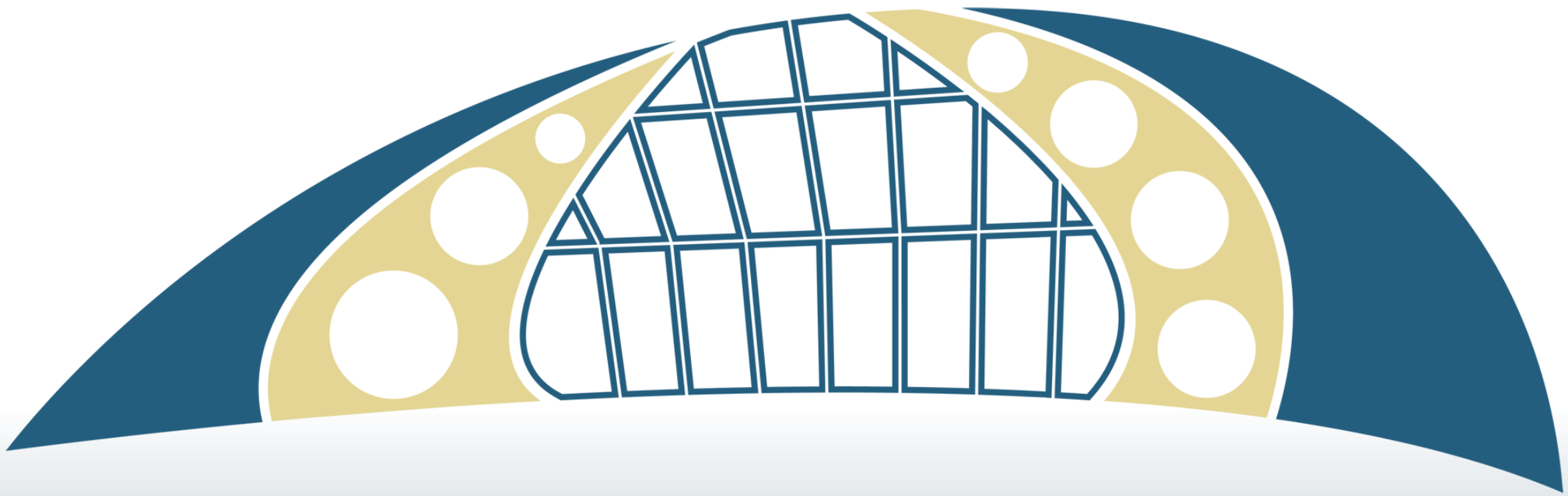
EXERCISE

HOW ARE WE GOING TO USE GIT?

- We will establish a project
- All code will be in the same repository
- Each team can make pull requests that focus on their components

MORE INFORMATION

- Version Control on Wikipedia
  - https://en.wikipedia.org/wiki/Version_control
- git tutorial on github
  - https://try.github.io/levels/1/challenges/1
- git tutorial on codecademy
  - https://www.codecademy.com/learn/learn-git



WESTMONT INSPIRED

COMPUTING LAB