Principles of Software Construction: Objects, Design, and Concurrency

Version Control

Christian Kästner Charlie Garrod

Learning Goals

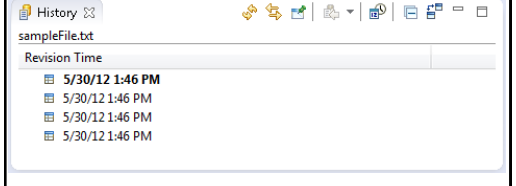
- Understand the benefits and limitations of version control (history, parallel development, branching, etc)
- Ability to cooperate on a code base using branching and merging; to resolve merge conflicts
- Distinguish classes of version control systems (local, central, distributed)

HISTORY

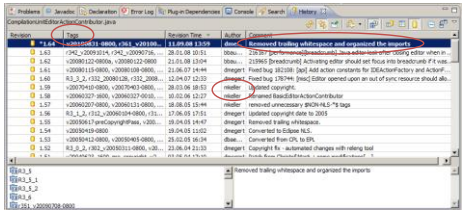
History

- Record relevant steps in the history of the project
- Supports partial undo
- Label cohesive development steps
  - Allows understanding changes
  - Explains rationale
  - Traceability to other development artifacts (bug trackers, CVEs, requirements, etc)
- Record who performed changes
- Implies a backup
- Early local systems: SCCS and RCS
- -> Version control systems useful even when used entirely locally

Eclipse's Local History

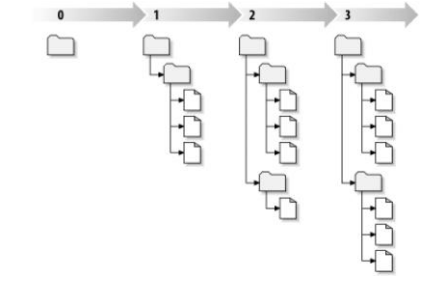


Revision History



Recording Rationale and Traceability

Versioning entire projects



Versioning and Tags

- Version numbers for files individually vs per project
- Version numbers vs hashes
- Tags to name specific states

While files to manage

- All code and noncode files
  - Java code
  - Build scripts
  - Documentation
- Exclude generated files (.class, …)
- Most version control systems have a mechanism to exclude files (e.g., .gitignore)

COLLABORATION

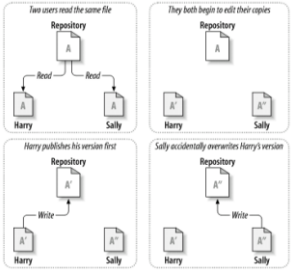
Collaborating on Files

- How to exchange files
  - Sends changes per email
  - Manual synchronization at project meeting
  - All files on shared network directory
- Permission models
  - Each file has an owner; only person allowed to change it
  - Everybody may change all files (collective ownership)

Concurrent Modifications

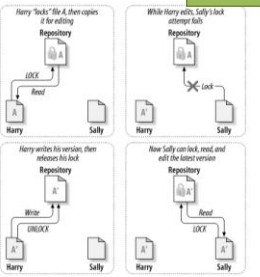
- Allowing concurrent modifications is challenging
- Conflicts (accidental overwriting) may occur
- Common strategies
  - Locking to change
  - Detecting conflicts (optimistic model)

Change Conflicts



Locking Files

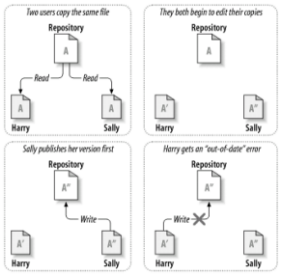
Practical problems of locking model?



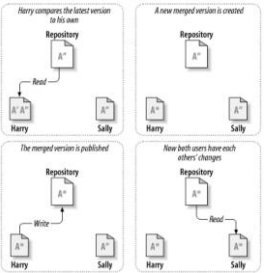
Locking Problems

- How to lock?
  - Central system vs announcement on mailing list
  - Forgetting to unlock common
- Unnecesary sequentializing
  - Cannot work on different concepts in same file
- False sense of security
  - Changing dependant files can cause conflicts not prevented by locking

Merging (1/2)



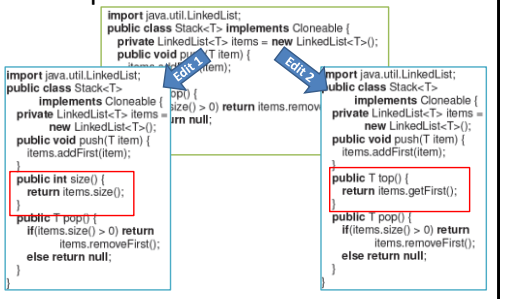
Merging (2/2)



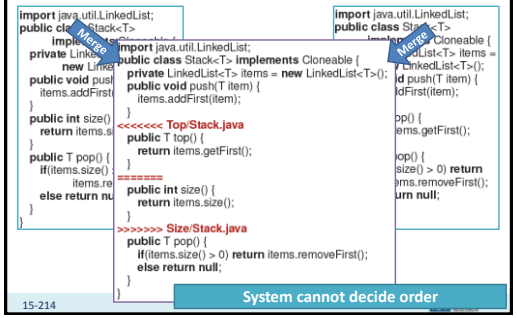
Example

```
import java.util.LinkedList;
public class Stack<T> implements Cloneable {
  private LinkedList<T> items = new LinkedList<T>();
  public void push(T item) {
    items.addFirst(item);
  }
  public T pop() {
    if(items.size() > 0) return items.removeFirst();
    else return null;
  }
}
```


Example



Example



3-way merge

- File changed in two ways
  - Overlapping changes -> conflicts
  - Merge combines non-conflicting changes from both
- Merging not always automatic
  - diff tool to show changes
  - Manual resolution of conflicts during merge (potentially requires additional communication)
- Automatic merge potentially dangerous
  -> syntactic notion of conflicts
- Merging of binary files difficult
- In practice: most merges are conflict free

BRANCHING

Branching

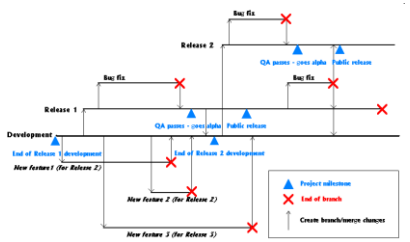
- Parallel copies of the source tree
- Can be changed independently, versioned separately, and merged later (or left separate)
- Often used for exploratory changes or to isolate development activities
- Many usage patterns, common:
  - Main branch for maintenance OR main development
  - New branches for experimental features; merge when successful
  - New branches for nontrivial maintenance work
  - Branches for maintenance of old versions

Branches (Verzweigen)

- Kopie des Quelltext
- Wird getrennt versioniert
- Kann wieder zusammengefügt werden (merge)

- Typisches Vorgehen
  - Hauptbranch für Wartung **oder** Hauptentwicklung
  - Neuer Branch für experimentelle Funktionalität, wird zusammengefügt wenn erfolgreich
  - Neuer Branch für Wartungsaufgaben
  - Teils neuer Branch für Varianten

Release management with branches



Variants and Revisions

- **Revision** replaces prior revision (temporal)
- **Variant** coexists with other variants
- **Version** describes both
- **Release:** Published and named version

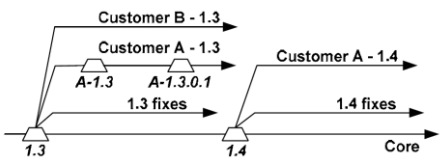
	V1.0	V1.1	V2.0	V3.0
Base system (Windows)	X	X	X	X
Linux variant		X	X	
Server variant			X	X
Extension for customer A		X	X	X
Extension for customer B				X

Semantic Versioning for Releases

- 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.
- Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

http://semver.org/

Variants and Revisions



[Staples&Hill, APSEC'04]

Managing variants

- Branching for variants does not scale well
- Requires special planning or tooling

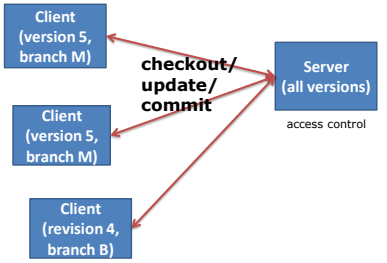
- Many solutions
  - Configuration files
  - Preprocessors
  - Build systems
  - DSLs
  - Software product lines
  - …

```
/* common parts */
...
/* dependent on operating system */
#if (OS == Unix)
...
#elif (OS == VMS)
...
#else
...
#endif
...
```

CENTRAL VERSION CONTROL (E.G., SVN)

Classes of version control systems

- Systems supporting merging and/or locking
- Local version control
  - Local history of files: SCCS (1970s), RCS (1982)
- Central version control
  - Versions stored on central master server
  - Clients synchronize with server (update, commit)
  - CVS (1990), SVN (2004), Perforce, Visual SourceSafe
- Distributed version control
  - Many repositories; synchronization among repositories (push, pull)
  - Git (2005), Mercurial, Bitkeeper, ClearCase



Typical work cycle

- Once: Create local workspace
  - svn checkout
- Update workspace:
  - svn update
- Perform changes in workspace:
  - svn add
  - svn delete
  - svn copy
  - svn move
- Show workspace changes:
  - svn status
  - svn diff
- Revert changes in workspace:
  - svn revert
- Update and merge conflicts:
  - svn update
  - svn resolved
- Push workspace changes to server:
  - svn commit

CVS vs. SVN

CVS

- Improvement over RCS in tracking entire directories
- Revision number per file
- Text files (binary files possible)

SVN

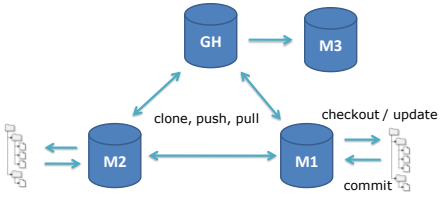
- Revision numbers for project
- Atomic commits (commiting multiple files at once)
- Tracking files and directories
- Support renaming
- Tracking of Metadata

DISTRIBUTED VERSION CONTROL (E.G., GIT)

Git

- Distributed version control
- No central server necessary (but possible)
- Local copies of repositories (containing all history)
  - Locally SVN like functionality: checkout, update, commit, branch, diff
- Nonlinear development: each local copy can evolve independently
- Synchronization among repositories (push/fetch/pull)
- Fast local operations (branch, commit, diff, ...)

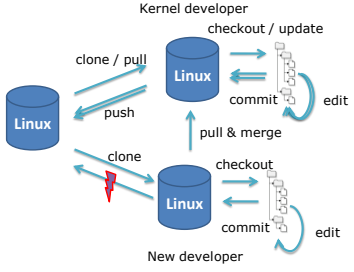
Overview



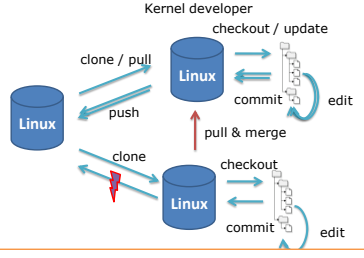
Distributed Versions

- Versions not globally coordinated/sorted
- Unique IDs through hashes, relationships tracked in successor graph
  - e.g., 52a0ff44aba8599f43a5d821c421af316cb7305
  - Possible to merge select changes (cherry picking)
  - Possible to rewrite the history as long as not shared remotely (git rebase etc)
- Cloning creates copy of repository (including all versions)
  - Tracks latest state when cloned, relevant for updating and merging
  - Normal checkout and commit operations locally
  - Commits don't change original repository
- Fetch and pull get missing versions from remote repository (one or more)
- Push operations sends local changes to remote repository (one or more), given access rights

Example workflow

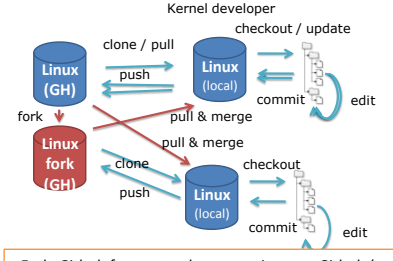


Pull Requests



Pull request: Github feature to ask developer to pull a specific change (alternative to sending email); integration with Travis CI

Forks



Fork: Github feature to clone repository on Github (own copy with full rights)

Forks

Kernel developer

checkout / update

Caution:
Please to not fork 214 repositories.

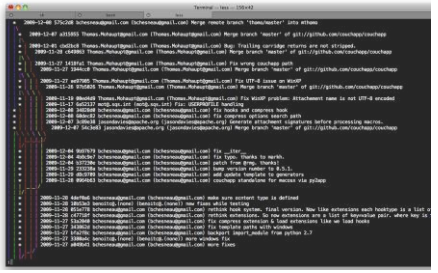
214 Collaboration Policy: "Here are some examples of behavior that are inappropriate: Making your work publicly available in a way that other students (current or future) can access your solutions, even if others' access is accidental or incidental to your goals."

(local) commit edit

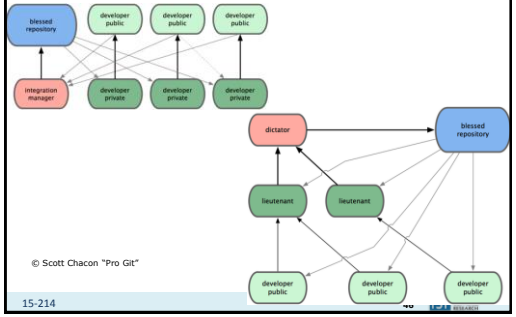
Fork: Github feature to clone repository on Github (own copy with full rights)

Repositories in mustache.js

Git History

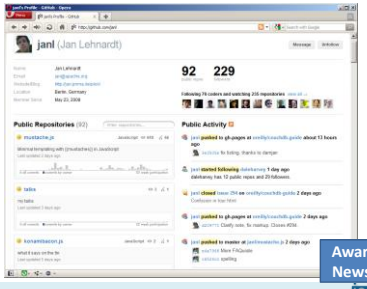


Git and Central Repositories



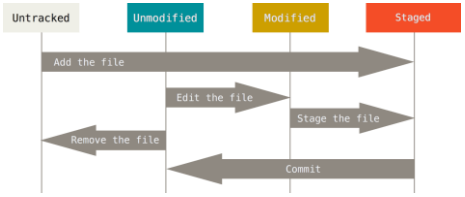
© Scott Chacon "Pro Git"

Social Coding



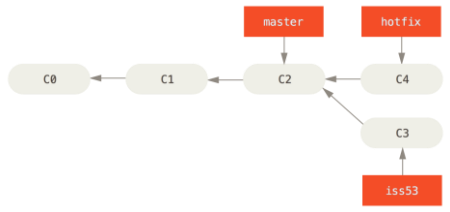
Awareness/ News Feeds

Git Internals



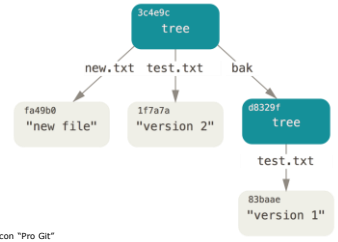
© Scott Chacon "Pro Git"

Git Internals



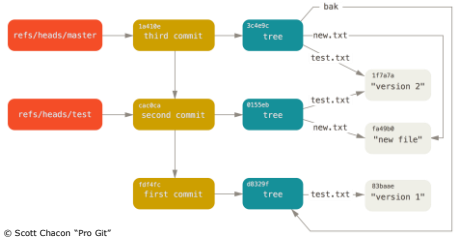
© Scott Chacon "Pro Git"

Git Internals



© Scott Chacon "Pro Git"

Git Internals



© Scott Chacon "Pro Git"

Summary

- Version control has many advantages
  - History, traceability, versioning
  - Collaborative and parallel development
- Locking vs merging and merge conflicts
- Collaboration with branches
- From local to central to distributed version control