On Version Control

Earl Barr
Department of Computer Science
University College London

March 2013

Programming Is Writing

- Programming is *writing* in a programming language.
- Like authors, programmers
  - read more often then they write; and
  - write for themselves and each other

Programming Is Writing

- Programming is *writing* in a programming language.
- Like authors, programmers
  - read more often then they write; and
  - write for themselves and each other
- Unlike authors, they're also writing instructions for a machine

A Cool Word

pa · limp · sest
/'pa-ləmp-sest/
Noun

1. A manuscript or piece of writing material on which the original writing has been effaced to make room for later writing.
2. Something reused or altered but still bearing visible traces of its earlier form.

The Archimedes Palimpsest

Source Code as Palimpset

Like parchment, source code is constantly being worked and reworked:

- refactored
- fixed
- extended

Source Code as Palimpset

Like parchment, source code is constantly being worked and reworked:

- refactored
- fixed
- extended

Change and rework is a measure of the vitality of software

Version Control

- Tracks the history of edits
- Allows reverting changes and cherry-picking
- Language independent
- Differs from back up
  - Granularity
  - Commit messages
  - Copying data onto other hardware is a side-effect

Infeasible before the computer age

Goals

- History of Version Control (VC)
- Indispensable tool
- Teach you about VC, focusing on Git
- Pose research questions

Prehistory

Entirely manual:

```
cp file file.timestamp
```

- Still heavily used!

Prehistory

Entirely manual:

```
cp file file.timestamp
```

- Still heavily used!
- Microsoft Word is a case in point, despite its revision history

Prehistory

Entirely manual:

```
cp file file.timestamp
```

- Still heavily used!
- Microsoft Word is a case in point, despite its revision history
- Closer to backup: fine-grained, but no easy way to add commit messages

Files, aka The Stone Age

- Source Code Control System (SCCS), 1972
- Revision Control System (RCS), 1982
- VAX/VMS, circa 1975
  - At each write, $\langle\text{filename}\rangle;\langle\text{version}\rangle; \rightarrow \langle\text{filename}\rangle;\langle\text{version} + 1\rangle;$

RCS and SCCS store delta, or changes, or commits in a subdirectory of the source directory.

Set of Files and Sharing, the late Stone Age

- Co-commit a set of files, or *changeset*, instead of just a file
  - In particular, directory trees
- Support for client-server, via NFS

Set of Files and Sharing, the late Stone Age

- Co-commit a set of files, or *changeset*, instead of just a file
  - In particular, directory trees
- Support for client-server, via NFS

Concurrent Versions System (CVS), 1986

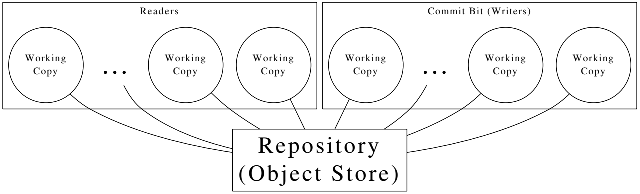
- Changeset not explicitly stored, but inferred from time window
- This time window is
  - narrow because the changeset commit is automatic, but
  - NOT ATOMIC!

Atomic Changesets and Sharing, aka The Bronze Age

SVN, 2000

- Changeset commited atomically and given a unique revision number
- Built in, native network support
- Perforce, Visual SourceSafe, and *many* others

Star Topology



- Working copy is distinct from the repository, which contains the object store.
- The "Commit bit"

Distribution, aka The Iron Age

- Co-commit sequences (*ordered* sets) of changesets aka branches
  - Totally ordered, within a branch
  - Partially ordered, across branches
- Everyone has a repository, with the complete history

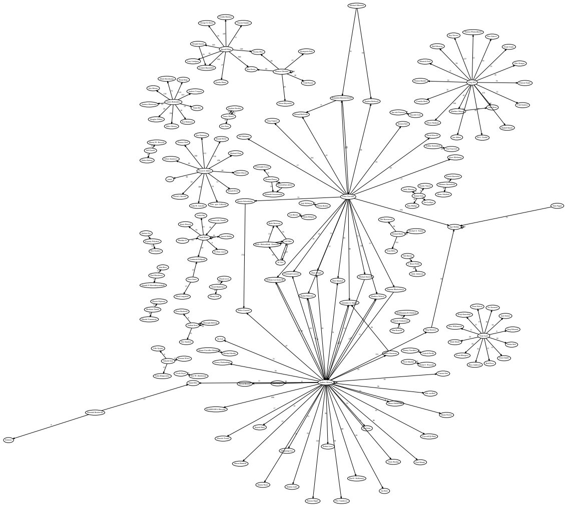
Distribution, aka The Iron Age

- Co-commit sequences (*ordered* sets) of changesets aka branches
  - Totally ordered, within a branch
  - Partially ordered, across branches
- Everyone has a repository, with the complete history

- BitKeeper 1998, used by Linux in 2000
- Git and Mercurial, 2005
- Bazaar (2005), GNU Arch (2006), Darcs, SVK, others. . .

The BitKeeper license for the Linux kernel was withdrawn amid claims that Andrew Tridgell had reverse-engineered the BitKeeper protocols.

Distribution, aka The Iron Age

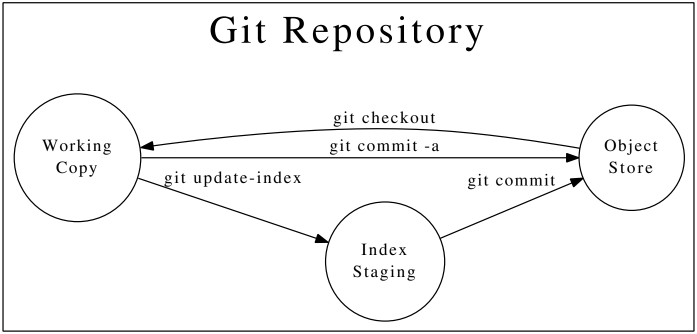


The Age of Aquarius?

The history of version control is one of increasing fidelity of what they record; from file, to sets of files (changesets) to, sequences of changesets, aka branches.
What will the next step be?

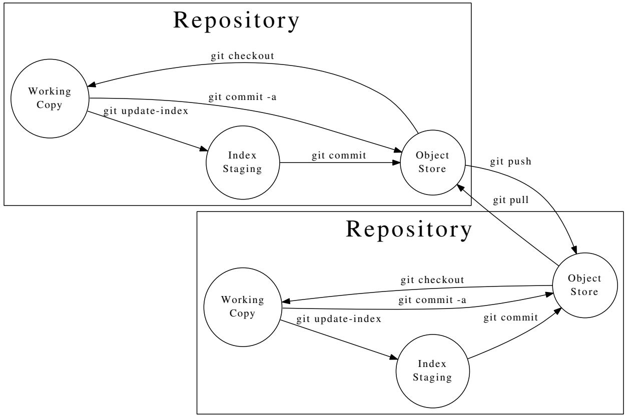
- One idea I have is language-awareness
- Better exploitation of history

All Repositories Are Created Equal



The object store is a database of commits; it is the history.

The Difference Distribution Makes



Must commit locally, before you can push remotely.

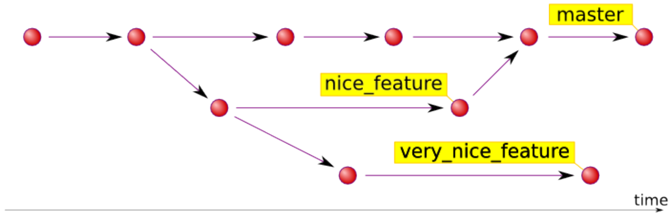
The Rapid Adoption of Distributed VC

- James Carville, 1992: “It’s the economy, stupid”

The Rapid Adoption of Distributed VC

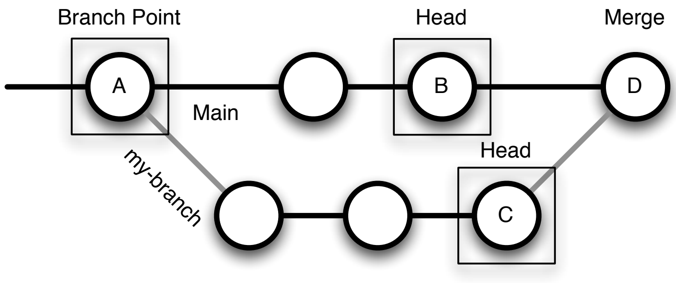
- James Carville, 1992: “It’s the economy, stupid”
- “It’s the branching, stupid”, *not* distribution
  - 60 projects
  - 1.5 $\rightarrow$ 3.7 branches / month
  - Retained star topology
- A Git object store *is* a set of branches

Branching



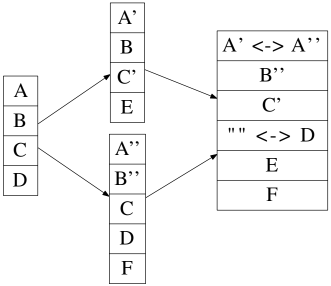
- Branches allow isolated work: they separate
  - debugging and new feature development from
  - integration, the handling concurrent work by others
  - Allow offline work, as when flying
- "Cohesive and Isolated Development with Branches" at FASE'12

3-way Merge



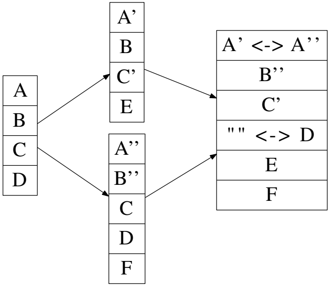
- Use common ancestor to identify blocks changed in neither, one or both
- At the time Git was written, SVN did not track the branch point and had no way to identify it

3-way Merge



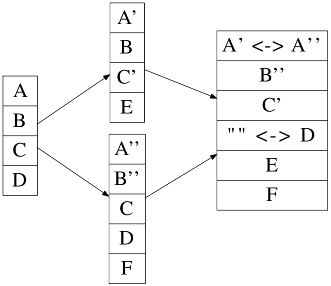
- Flag those changed in both as conflicts

3-way Merge



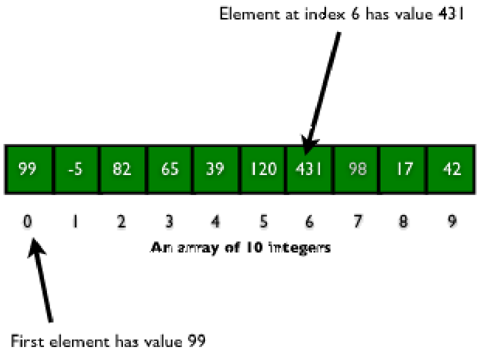
- Flag those changed in both as conflicts
- "I’m an egotistical bastard, and I name all my projects after myself. First ’Linux’, now ’git’." —Linus Torvalds

3-way Merge



- Research Idea: Use dataflow to find potential conflicts through globals.
    - Still a "stupid git" approach: notifies the user about changes in one or the other

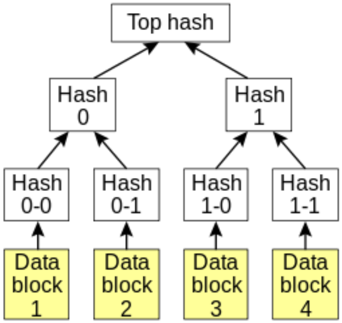
Location Addressable Storage



Think array, notably the memory array.

Content-Addressable Storage

- Associative array or hash table
- Use insecure hash function to identify a changeset
- This is how each repository generate universal keys, without communication.
- Git better demonstrates Linus Torvald's design prowess than Linux



The Git UI

- What I Hate About Git — Hacker News
- Reddit: A Nightmage of Mixed Metaphors
- Why Git sucks and you'll use it anyways

Daily Use

Action	SVN	Git
Create Repo	svnadmin create ⟨*repo*⟩	git init
Commit	svn commit -m "Fix Bug" file	git commit -am "Fix Bug"
Diff	svn diff	git diff
Status	svn status	git status
Restore	svn revert	git checkout
add, rm, mv	svn {add, rm, mv}	git {add, rm, mv}
History	svn log	git log
Remote	svn checkout ⟨*url*⟩	git clone ⟨*url*⟩
...	...	...

`http://git.or.cz/course/svn.html`

Advanced Commands

- git bisect: provide a property test, then binary search history
- git rebase: edit history

VC and Empirical Software Engineering

- Data is usually expensive
  - Countercurrent in the age of big data
  - Physics depends on colliders that cost billions
  - The core ESE difficulty
- VC has given SE
  - A cheap source of plentiful data
  - Perhaps also a street light

Continuous Integration

Continuous integration (CI) is the practice, in software engineering, of merging all developer workspaces with a shared mainline several times a day.

- Originated in extreme programming (XP)
- Seeks to prevent "integration hell"
- Adopted outside of XP
- Jenkins, IBM's Jira

CI build servers run tests

- upon each commit
- at a scheduled time
- after *n* commits

Research Questions

- What is the optimal granularity?
- Does that granularity vary with project?

Workflow and Promotion

A workflow is a set of conventions used to organize development.

- How is work divided up into tasks and assigned?
- How are the tasks integrated, *i.e.* promoted to main or deployed?

Workflow and Promotion

A workflow is a set of conventions used to organize development.

- How is work divided up into tasks and assigned?
- How are the tasks integrated, *i.e.* promoted to main or deployed?
  - Is it a free-for-all?

Workflow and Promotion

A workflow is a set of conventions used to organize development.

- How is work divided up into tasks and assigned?
- How are the tasks integrated, *i.e.* promoted to main or deployed?
  - Is it a free-for-all?
  - Is testing required?
  - Is review required?

Not knowing an organization's workflow can get you fired!

My Workflow

1. Always work in a branch other than main!
2. Commit whenever

My Workflow

1. Always work in a branch other than main!
2. Commit whenever
    - Caffeine run
    - Itchy nose
    - You feel like it

My Workflow

1. Always work in a branch other than main!
2. Commit whenever; this gives
    - Fine-grained backup
    - Frees me from thinking about *beautiful* commits
    - However, ugly commits write an UGLY history

My Workflow

1. Always work in a branch other than main!
2. Commit whenever; this gives
    - Fine-grained backup
    - Frees me from thinking about *beautiful* commits
    - However, ugly commits write an UGLY history
3. When ready to integration/promote, rebase

Rebasing Considered Harmful?

- Google “git rebase evil harmful”
- Public vs private branches

Rebasing

- For good, clean, understandable commits
  - Clear, pertinent commits messages
  - The project *builds*
  - Conceptually cohesive: fusing and breaking commits
- Just as branches separate integration and development, rebasing separates development from the concern of version history

My Workflow

1. Always work in a branch other than main!
2. Commit whenever
3. When ready to integration/promote, rebase
4. When on a team, ask someone else to review prior to promotion
    - Fresh eyes

¿The One True Way?

"The unexamined life is not worth living for a human being."
—Socrates, Apology

Commit Messages

- What to write?
- It's the documentation problem redux.
  - The Missing Links: Bugs and Bug-fix Commits
- ¡Why not what!

Indispensible Tool Redux

- git bisect
- DVC gives fine-grained, semantic back-up nearly for free
- Good commit messages help comprehension and navigation;
- Commits define sets of files and methods whose co-commit can help navigation.
- Continuous Integration
- Code Review Support — Gerrit

Git Resources

- Wikipedia, of course
- For you poor misguided Windows users, there is TortoiseGit
- Remember to upload your source and changes, not your keys or passwords
  - Slashdot: Github Kills Search After Hundreds of Private Keys Exposed

¿Questions?