Version Control Systems (Part I)

Devin J. Pohly <djpohly@cse.psu.edu>
Version control systems

A *version control system* is a system for keeping track of the changes made to a document (or collection of documents) over time.

- Any kind of document...
  - Resumes
  - TPS reports
  - Source code
Why?

• It’s a time machine!
  ‣ Look at old versions
  ‣ Never lose anything
  ‣ Revert your mistakes
  ‣ Code fearlessly!

• Collaboration
  ‣ Work in parallel
  ‣ Merge changes
  ‣ Social coding
In other words

UNTARS STARTER CODE

OVER FINISHED CODE
A word about words

• These terms all refer to the same thing:
  ‣ Version control system
  ‣ Revision control system
  ‣ Source code/control management
  ‣ VCS/RCS/SCM

• For this lecture: “version control” and VCS
The original

- Printing press: 1440
- Book editions
  - Edition numbers
  - Copyright dates
- No visible record of what changed
Basic concepts

- **Revision**: one meaningful change or set of changes
- **Repository**: where all of the revisions are stored
- **Working copy**: where the user makes changes
- **Check out**: copy one revision from repository to working copy
- **Check in/commit**: add a new revision to repository
Basic concepts

- **Branches**: parallel lines of development
- **Trunk/master**: main development branch
- **Tip/head**: latest revision on a branch
- **Tag**: special name given to an important revision
  - Often used for numbered releases like “v3.14”
The first generation

- Local VCSes
- 1970s and 80s
- SCCS, RCS
- Repository stored in a shared local directory
- User must lock a file before making changes
- Lock-edit-unlock model
RCS usage

• Extremely simple
• Check out (read-only)
  ‣ rcs co foo.h
• Check out and lock
  ‣ rcs co -l foo.c
• Check in and commit changes
  ‣ rcs ci foo.c
How RCS works

- Each file has its own repository in the RCS directory, in which all of that file’s revisions are stored.
How RCS works

Anyone can check out a read-only working copy of a file.
How RCS works

• If Alice wants to make changes to foo.c, she must lock the file for writing when she checks it out.

rcs co -l foo.c

Alice

foo.c 1.12

foo.h 1.5

Repository

foo.c 1.1–1.12

foo.h 1.1–1.5

main.c 1.1–1.50

Bob

foo.h 1.5

foo.c
How RCS works

- Since Alice has locked foo.c, nobody else can lock it.
- Alice can now safely edit her local copy of the file.
How RCS works

- You can only commit changes to a file if you hold the lock.
- Committing foo.c checks in Alice’s changes as a new revision, then *unlocks* the file so others can lock it.
How RCS works

• When Alice commits her modified foo.c, the repository creates the new *revision number* 1.13.
The second generation

- Centralized VCSes
- 1980s to 2000s
- CVS, **Subversion** (SVN)
- Still widely used
- Repository on a server with many clients
- Copy-modify-merge model
How Subversion works

- Spot the differences!
How Subversion works

- Files are stored in one repository rather than individual ones.
- Repository and users can all be on different hosts.
How Subversion works

svn checkout URL

Alice

foo.c

foo.h

main.c

Repository
1–56

foo.c

foo.h

main.c

svn checkout URL

Bob

foo.c

foo.h

main.c

• Checkout still does the same thing: gets a working copy of the latest revision (all files) from the repository.
• You don’t have to lock files to change them.
How Subversion works

- If Alice *modifies* some files and commits her changes...
How Subversion works

- ... a new revision of the repository (r57) is created.
How Subversion works

- If Bob makes changes that don’t overlap with Alice’s, Subversion can *merge* them automatically.
- This is what happens most of the time.
How Subversion works

• If Subversion can’t merge the changes automatically, it notifies Bob that there is a *merge conflict*. 
How Subversion works

- So Bob *updates* his repository with Alice’s changes, merges them with his, and tries to commit again.
How Subversion works

- Bob’s commit succeeds this time, and the repository is now at revision 58.
Commit graphs

- By convention, the arrow points from the child revision to the parent revision.
- Every branch in Subversion or RCS has an entirely linear commit graph.
- (Branches are linearized when you merge them.)
The third generation

• Distributed VCSes
• 2000s to today
• Bazaar, Git, Mercurial
• Seeing widespread use
• Everyone has a full repository
• Highly collaborative
  ‣ Linux development
  ‣ GitHub and other “social coding” sites
Sharing your commits

• When you commit your changes, the revision is stored in your local repository

• All communication is between repositories
  ‣ You *push* local commits to a remote repository...
  ‣ and you *pull* commits from a remote repository into the local one.
How Git works

- Spot the differences
How Git works

- Alice and Bob will both have their own repositories, no different from the “official” one.
How Git works

- First step is to **clone** the repository, not check it out.
- This gives you a local clone of the entire repo!
How Git works

- Alice can now work locally (and offline), without worrying about other repositories.
How Git works

- Alice edits `foo.c` as usual, adds the changes to her commit, and commits.
How Git works

- Revision B, based on A, is now in Alice’s repository.
How Git works

- In our Git example, Alice has committed revision B "onto" revision A.
How Git works

- In our Git example, Alice has committed revision B “onto” revision A.
- She can then commit another revision C onto that.
Push and pull in Git

- Alice can *push* her new commits to another repository, such as the “official” one.
- The commit being pushed must be a *descendant* of the remote one.

```bash
git push
```

- **Alice’s repository**: ..C
  - foo.c
  - foo.h
  - main.c

- **“Official” repository**: ..A
  - foo.c
  - foo.h
  - main.c

- **Bob’s repository**: ..B'
  - foo.c
  - foo.h
  - main.c
Push and pull in Git

- The official repository now contains Alice’s commits.
- Notice Bob has also committed B' but not yet pushed!
Push and pull in Git

Bob’s commit graph has his new revision, but none of Alice’s.
Push and pull in Git

- Bob cannot push his changes yet, because B' is not a descendant of C.
Push and pull in Git

• Bob needs to get Alice’s new changes and merge them.
• First he *pulls* the changes from the official repo...
• ... and they are added to his repository.

• He can now *merge* Alice’s changes with his.
Git merges

- This creates a new *merge revision* D which is a child of both B' and C.
Coming full circle

- Since D is a descendant of C, Bob can now push!

```bash
git push
```

Alice

- Alice’s repository
  - foo.c
  - foo.h
  - main.c

“Official” repository

- foo.c
- foo.h
- main.c

Bob

- Bob’s repository
  - foo.c
  - foo.h
  - main.c
Coming full circle

- The official repository now has revision D, which contains both Alice’s and Bob’s changes.
Coming full circle

- Developers can also collaborate directly.
- Here Alice gets Bob’s latest commits from Bob himself instead of from the “official” repository.

```
git pull BOB_URL
```
Coming full circle

- This could be used, for instance, to collaborate on experimental features that aren’t ready for prime-time.
Trends in version control

• Isolated to collaborative
• Serial to concurrent
• Linear to branching
• Centralized to distributed
• Limited workflows to many possibilities
Next time

• It’s Hands-On Friday on Wednesday!! WHAT IS THE WORLD COMING TO??

• But seriously, we’re learning Git.