CS 307: Software Engineering

Lecture 4: Version Control

Prof. Jeff Turkstra
Announcements

- Teams are assigned
  - Email me immediately if there is a problem
- Consultation meeting
  - Thursday, January 19, 9:30am – 5:30pm
  - LWSN 3162
  - You should have chosen a slot already
- Project Charter due
  - Friday, January 20 3:00pm
Assignment

- Read “Software Engineering: Working As A Team” before Friday’s lecture
  - It’s on the website
Lecture 04

- Version control
- git
- github
Revision control

- Or version control is a mechanism for managing changes made to files
  - Source code
  - Documentation
  - Binary files
    - At least not usually
- Think of book editions
- Modern systems support collaboration
Version control systems

- RCS
  - Developed here at Purdue!
- CVS
- SVN
- Perforce
- Git
- Mercurial
- Bazaar
- More!
Centralized model

- Single authoritative repository
- All code checked in/out from this location
- Single source of version history
- Local changes are not “versioned”
Distributed model

- No authoritative copy, only working copies
- Each repository (copy) contains all branches and revision history
  - Initial clone can be slow
- Common operations are fast
  - No need to talk to central server
- Communication only occurs when sharing changes
- Push/pull changes
git

- Created by Linus Torvalds
  - For Linux kernel development
- Replacement for BitKeeper
  - Proprietary system with politics
- Junio Hamano is now the project maintainer
Resources

- **Git website:** https://git-scm.com/
- **Git for Computer Scientists**
  - http://eagain.net/articles/git-for-computer-scientists/
- **Git for Scientists: A Tutorial**
  - http://nyucll.org/pages/GitTutorial/
Snapshots vs deltas
Getting started

- git config --global user.name "Joe Smith"
- git config --global user.email "joe@smith.com"
- git config --list
Create a repository

- Two ways: `git init` and `git init --bare`

- Working repository
  - Root contains checked out copy of the repository
  - and a `.git` subdirectory with the revision history

- Bare is for sharing
  - No working copy
  - e.g., `github.com`
Local flow

- HEAD
- Index
- Working Directory

1. Checkout the project
2. Stage files
3. Commit
Commands

- `git status`
  - View the state of files in the current directory
- `git add`
  - Stage a file for commit (add to index)
- `git rm`
  - Remove files from the working tree and index
- `git commit`
  - Record changes in the repository
- `git diff`
  - Display differences between commits, working tree, etc
- `git log`
  - View the commit log and comments
Undoing a commit

- The nice way
  - `git revert <hash>`

- The hard way
  - `git reset --hard HEAD~1`
  - `git reset --soft HEAD~1`
    - Don’t do this if you already pushed

- There are other ways, but you need to be sure nobody has pulled

- Recover lost commits
  - `git checkout -b newbranch <hash>`

- Unless you’ve run `git-gc`
Git Data Transport Commands

http://osteele.com

- commit -a
- add (-u) → commit
- push
- pull or rebase
- checkout HEAD
- fetch
- revert
- checkout
- diff HEAD
- diff
Branching

- “Branch early, branch often”
- Branches are cheap
- Commits are cheap
- `git branch branchname`
- `git checkout branchname`
- Or `git checkout -b branchname`
- There is always a master branch
Merging

- Checkout the branch you want to merge onto
- Merge the other branch
  - Resolve any conflicts
    - Re-commit
- Done!
Tagging

- `git tag <tagname> <hash>`
  - Lightweight tag, simply a file pointing to the hash
- `git tag -a <tagname> <hash>`
  - `-s` to sign
  - Annotated tags
  - `git show <tagname>`
- Have to explicitly push tags to remote
  - `git push origin <tagname>`
  - `git push --tags`
Demo
Working with remote

- `git fetch`
  - Obtain remote changes

- `git merge`

- `...or git pull`
  - Fetches and merges

- `git push`
  - Propagate your changes somewhere else
Protocols

- **Local file system**
  - Simple
  - Requires all users to use the same system (or at least, shared file system)

- **Git protocol**
  - Fastest
  - Lack of authentication
  - Anonymous public access
  - Harder to set up, port 9418
- **SSH**
  - Secure, authenticated access
  - No anonymous access

- **HTTP(S)**
  - Easy to set up
  - Unlikely to be blocked
  - Inefficient
  - Both authenticated and public access
Demo
Multiple remotes

- Contributors and developers are often in a hierarchy
- SSH write access for core developers
- Public read access
- Core developers (“head coders”)
  - Pull from other contributors’ repositories
  - Merge as needed
  - Push to the public repository
Pull request

- Code is *merged* based on trust
- Contributor requests the maintainer pull their source code change
  - Maybe after maintainer asked for something to be done, maybe not
- Discussion associated with request
  - Opportunity for inspection and code review
- Request can be accepted or rejected
Rebasing

- Similar to merging, but creates a linear history
- Can help commits apply cleanly on a remote branch
- Work in a branch, rebase onto origin/master
- Rebasing replays changes from one line of work onto another, in order
**No**

- **Do not rebase commits that exist outside of your repository**
- Rebasings rewrites history and removes commits
- Someone else may have already based something on one of those commits
Rebase vs. merge

- Merges preserve history
  - Records what actually happened
- Rebase rewrites and erases history
  - “Tell a story”
- Never rebase anything that you have pushed somewhere else
Creating a remote branch

- `git branch -b newbranch`
- `git push <remote> newbranch`
- ...or maybe `git push -u`
  - Sets up an upstream branch
- To delete, `git push <remote> :newbranch`
Cherry picking

- Checkout a branch
- Look at the log
- Create a new branch
  - Usually
- `git cherry-pick <hash>`
Dealing with bugs

- **Bisecting**
  - `git bisect start`
  - `git bisect bad`
  - `git bisect good <tag or hash>`
  - Then mark each one good or bad (or skip)
  - `git bisect reset`

- **Binary search, not linear**
- **git blame**
Demo
Questions?