Announcements

- Teams and coordinators are assigned
  - Email me immediately if there is a problem
- Project Charter due on Friday, September 1 at 3:00 PM
  - Your job to set up meeting with project coordinator to review draft
- Homework 1 academic honesty
- VW Engineer
Assignment

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

- Version control
- git
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/](https://git-scm.com/)
- **Git for Computer Scientists**
  - [http://eagain.net/articles/git-for-computer-scientists/](http://eagain.net/articles/git-for-computer-scientists/)
- **Git for Scientists: A Tutorial**
- [http://nyuccl.org/pages/GitTutorial/](http://nyuccl.org/pages/GitTutorial/)

```bash
$ git help ...
$ man git
```
Snapshots vs deltas
git hashes

- Everything in git is check-summed
  - Corruption is detected
  - Some malicious activities can be detected
  - SHA-1
    ef827df18863c71e70802c9be2310be0cb223f00

- References to a particular commit or file can be made by hash
Getting started

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

git config --global core.editor vi
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`
Clone

- Step number one
  git clone repo workdir

- This creates a complete copy of the repository locally in workdir/.git

- Along with a checked out copy of the latest master branch in workdir
Working directory

- Working directory or tree refers to the currently checked out version of the project
- Sourced from the repository
- This is where you work
Tracked vs Untracked

- Tracked files are in the last snapshot
  - Unmodified, modified, staged
- Untracked is everything else
- All files are tracked and unmodified in an initial clone
Tracked States

- **Committed**
  - Exists in your local repository (.git)
  - Usually pointed to by **HEAD**

- **Staged**
  - Marked for the next commit
  - Stored in the **Index**

- **Modified**
  - File has changed, but is not committed nor marked
Local flow

- Think of Index as a staging area
- HEAD is stored in the repository
git status

- Shows the state of the working tree
  - Differences between index and HEAD
    - What you would commit if you ran `git commit`
  - Differences between working tree and index
    - What you could commit if you ran `git add` first
- Untracked paths
  - What you could commit if you ran `git add` first
Demo

$ mkdir myrepo.git; cd myrepo.git
$ git init --bare
$ cd ..
$ git clone myrepo.git mywd
$ cd mywd; echo "This is file 1" > file1
$ git status

...and github.com
git add

- Begin tracking a new file
- Stage files for commit that have been modified
- Mark merge-conflicted files as resolved
- Adds the file to the Index
Demo

$ git add file1
$ git status
$ echo "Hmm" >> file1
$ git status
$ git rm --cached file1
$ git commit file1
$ git push  # To create master
$ git pull
Undoing

$ git status
$ echo "asdf" >> file1
$ git status
$ git add file1
$ git reset HEAD ...
$ git status
$ git checkout ...
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/or 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
$ git diff
$ echo "1234" >> file1
$ git diff
$ git diff master
$ git diff --staged
git commit

- Moves Index to repository

  $ git commit -a
  
  - git add; git commit
  
  - $ git log

- $ git commit --amend
  
  - Always uses Index
  
  - No files? (Only) changes commit message
Demo

$ git add file1
$ git commit
$ git log
# Change file1
$ git commit --amend file1
$ git log
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
  - $ git branch -d branchname
  - To delete
Demo

$ git branch -b newfeature
$ git branch
$ git branch -a
$ git checkout master

- Updates working tree to match Index or specified tree
- *Local changes are kept*
Merging

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

$ git clone myrepo.git joewd
$ cd joewd # do stuff
$ cd ../mywd
$ git remote -v
$ git remote add joe ../joewd
$ git remote show joe
Working with remote

- `git fetch`
  - Obtain remote changes

- `git merge`

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

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

- Two users

```bash
$ git checkout -b feature2
$ git push joe feature2
$ git pull joe feature2
$ git branch --set-upstream-to=joe/feature2 feature2

$ git remote show joe
```

```bash
$ git remote show joe
```
Git Data Transport Commands

http://oastele.com

commit -a

add (-u)  commit

push

workspace  index  local repository  remote repository

pull or rebase

fetch

revert

checkout HEAD

checkout

compare

diff HEAD

diff
Creating a remote branch

- `git remote (-v)`
- `git branch -b newbranch`
- `git push <remote> newbranch`
- ...
or maybe `git push -u`
  - Sets up an upstream branch
- **To delete,**
  `git push <remote> :newbranch`
The other direction

$ git checkout -b bname remote/branch

Or

$ git checkout --track remote/branch
Undoing a commit

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

- **The hard way**
  
  git reset --hard HEAD~1
  git reset --mixed 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
Demo
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
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
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
Demo
No

- Do not rebase commits that exist outside of your repository
- Rebasing 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
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 filename**
Questions?