Git: Distributed Version Control

Lecture 3
What Does "D" Stand For?

- Distributed version control
  - Multiple people, distributed across network

- Each person has their own repository!
  - Everyone has their own store (history)!
  - Big difference with older VCS (eg SVN)

- Units of data movement: changesets!
  - Communication between teammates is to bring stores in sync
  - Basic operators: fetch and push
Sarah's Repository

HEAD
master

wt

a ← b ← c ← d ← e

Sarah
And Matt's Repository

Sarah

Matt

HEAD
master

wt

wt
Some Shared History
Fetch: Remote Store → Local

sarah$ git fetch mt

new changesets added to store

remote branch

HEAD
master

working tree unaffected!

mt/master

wt
Remote Repository Unchanged
Workflow: Merge After Fetch

sarah$ git merge mt/master
Remote Repository Unchanged
View of DAG with All Branches

$ git log --oneline --graph --decorate --all

* 1618849 (HEAD, origin/master, master) clean up css
*  d579fa2 (alert) merge in improvements from master
|\|
| * 0f10869 replace image-url helper in css
| *  b595b10 (origin/alert) add buckeye alert notes
| *  a6e8eb3 add raw buckeye alert download
|/
* b4e201c wrap osu layout around content
*  e9d3686 add Rakefile and refactor schedule loop
*  515aaa3 create README.md
*  eb26605 initial commit
Your Turn

- Show the state of Matt's repository after each of the following steps
  - Fetch (from Sarah)
  - Merge
Matt’s Repository after Fetch

matt$ git fetch sr
Sarah and Matt's Repositories

Sarah

Matt

HEAD
master

HEAD
mt/master
Some Shared History

Sarah

Matt

HEAD master

mt/master

HEAD master
Matt’s Repository after Merge

matt$ git merge sr/master
Pull: Fetch then Merge

- A "pull" combines both fetch & merge
  ```
  matt$ git pull sr
  ```
- Advice: Prefer explicit fetch, merge
  - After fetch, examine new work
    ```
    $ git log --all #see commit messages
    $ git checkout #see work
    $ git diff #compare
    ```
  - Then merge
  - Easier to adopt more complex workflows (e.g., rebasing instead of merging)
Push: Local Store $\rightarrow$ Remote

- Push sends local commits to remote store
- Usually push one branch (at a time)
  
  sarah$ git push mt fix
  
  - Advances Matt's fix branch
  - Advances Sarah's mt/fix remote branch

- Requires:
  1. Matt's fix branch must not be his HEAD
  2. Matt's fix branch must be ancestor of Sarah's

- Common practices:
  1. Only push to bare repositories (ie no working tree, so no HEAD)
  2. Get remote store's branch into local DAG (ie fetch, merge, commit) before pushing
Remote's Branch is Ancestor

Sarah

Matt

HEAD

mt/fix

fix

HEAD

master

fix

wt
Push: Local Store → Remote

sarah$ git push mt fix

Matt

HEAD
master

HEAD
fix

mt/fix

working
tree
unaffected!

wt
Commit/Checkout vs Push/Fetch

Local
- working directory
- staging area
- local repo

Remote
- remote repo

Flow:
- `git add`
- `git commit`
- `git push`
- `git fetch`
- `git checkout`
- `git merge`
Common Topology: Star

- Team has $n+1$ repositories
  - 1 shared central repository (bare)
  - 1 local repository / developer
- Each developer *clones* central repository
  - Cloning creates a remote called "origin"
  - Default source/destination for fetch/push
- Variations for central repository:
  - Everyone can read and write (ie push)
  - Everyone can read, but only 1 person can write (responsible for pulling and merging)
Workflow: Configure Git

- Each team member, in their own VM
  - Set identity for authoring commits
    
    $ git config --global user.name "Brutus Buckeye"
    
    $ git config --global user.email bb@osu.edu

  - Optional: diff and merge tool (eg meld)
    
    $ sudo apt-get install meld # to get tool
    
    $ git config --global merge.tool meld
    
    $ git config --global diff.tool meld
    
    # example use:
    
    $ git difftool e9d36
Workflow: Initialize Central Rep

- One person, once (ssh'ed to stdlinux):
  - Create central repository in group's project directory (/project/c3901aa03)
    
    $ cd /project/c3901aa03
    $ mkdir rep.git  # ordinary directory
    
    - Initialize central repository as bare and shared within the group
      
      $ git init --bare --shared rep.git

- Note: Hosting services (e.g., GitHub, BitBucket) often have a web interface for this step
Workflow: Initialize Repository

- Each team member, once, in their VM
  - Create local repository by *cloning* the central repository
    
    $ git clone
    ssh://brutus@stdlinux.cse.ohio-state.edu//project/c3901aa03/rep.git
    mywork

  - You will be prompted for your (stdlinux) password (every time you fetch and push too)
  - To avoid having to enter your password each time, create an ssh key-pair (see VM setup instructions)
Ignoring Files from Working Tree

- Use a `.gitignore` file in root of project
  
  ```
  # Ignore auto-saved emacs files
  *~
  
  # Ignore bundler config
  ./bundle
  
  # Ignore the default SQLite database
  /db/*.sqlite3
  
  # Ignore all logfiles and tempfiles
  /log/*.log
  /tmp
  ```
Workflow: Local Development

Each team member repeats:
- Edit and commit (to local repository) often
  $ git status/add/rm/commit
- Pull others' work when can benefit
  $ git fetch origin # bring in changes
  $ git log/checkout # examine new work
  $ git merge, commit # merge work
- Push to central repository when confident
  $ git push origin master # share
Git Clients and Hosting Services

- Recommended client: Command line!
- Various GUIs:
  - Linux: gitg, git-gui, git-cola, giggle
  - Win/mac GUI: SourceTree
- Lots of sites for hosting your repos:
  - GitHub, Bitbucket, SourceForge, Google Code,...
- These cloud services provide
  - Storage space
  - Pretty web interface
  - Issues, bug tracking
  - Workflow with "forks" and "pull requests" to promote contributions from others
Warning: Academic Misconduct

- GitHub is a very popular service
  - But only *public* repo's are free
  - Edu discount gives you 5 private repo's
  - 3901 has an account for creating more private repo's (see class web site)

- Bitbucket has free private repo's, for small teams (< 5 collaborators)

- Public repo's containing coursework can create academic misconduct issues
  - Problems for poster
  - Problems for plagiarist
Summary

- Push/fetch to share your store with remote repositories
  - Neither working tree is affected
- Branches in history are easy to form
  - Committing when HEAD is not a leaf
  - Fetching work based on earlier commit
- Advice
  - Learn by using the command line
  - Beware academic misconduct
Group Formation

- Groups of 4 (or 5, but 4 is better)
- Exchange contact information
- Each person choose a primary technical area:
  - HTML/CSS
  - JavaScript
  - Ruby
- Group constraints on choices:
  - Each technology must be represented
  - No more than 2 people per technology
- Choose a secondary interest as well
  - “Don’t Care” is fine (as primary or secondary)
Mercurial (hg): Another DVCS

- Slightly simpler mental model
- Some differences in terminology
  - git fetch/pull ≈ hg pull/fetch
  - git checkout ≈ hg update
- Some (minor) differences in features
  - No rebasing (only merging)
  - No octopus merge (#parents ≤ 2)
- But key ideas are identical
  - Repository = working directory + store
  - Send/Receive changes between stores
Advanced: Undoing Mistakes

- Say you want to throw away all your uncommitted work
  - Ie "Roll back" to last committed state
- Checkout won't work!
Reset: Discarding Changes

$ git reset --hard
$ git clean --dry-run # list untracked files
$ git clean --force # remove untracked files
Reset: Discarding Commits

$ git reset --hard HEAD~1
# no need to git clean, since wt was already clean

HEAD (and attached branch) moved

now unreachable