
CSE 374

Programming Concepts & Tools

Hal Perkins

Winter 2017

Lecture 16 – Version control and git

Administrivia

- HW6 writeup posted shortly – please read before Fri.
 - Part I due in a week – skeleton code + makefile
 - Full assignment due the following week
 - Detailed discussion in next couple of classes
- Remember to send in partner info by 11 pm tonight
 - Will create gitlab repositories for groups after that
 - Watch for email from gitlab.cs.washington.edu with repo information and link, then log on, set up keys, clone repo, and try it out
 - Anyone still need a partner?

Where we are

- Learning tools and concepts relevant to multi-file, multi-person, multi-platform, multi-month projects
- Today: Managing source code
 - Reliable backup of hard-to-replace information (i.e., sources)
 - Tools for managing concurrent and potentially conflicting changes from multiple people
 - Ability to retrieve previous versions
- Note: None of this has anything to do with code. Like make, version-control systems are typically not language-specific.
 - Many people use version control systems for everything they do (code, papers, slides, letters, drawings, pictures, . . .)
 - Traditional systems were best at text files (comparing differences, etc.); newer ones work fine with others too
 - But be sure to check before storing videos & other media

Version-control systems

- There are plenty: sccs, rcs, cvs (mostly historical); subversion, git, mercurial, perforce, sourcesafe, ...
- Terminology and commands aren't particularly standard, but once you know one, the others aren't difficult – the basic concepts are the same
- svn still used – single central repository, users have individual working copies
- git and mercurial: distributed version control
 - Same core ideas as svn, but every user has a full copy of the repository; allows easy branching & merging for large collaborations (e.g., linux kernel)
 - We'll use git, widely used these days

What is git?

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

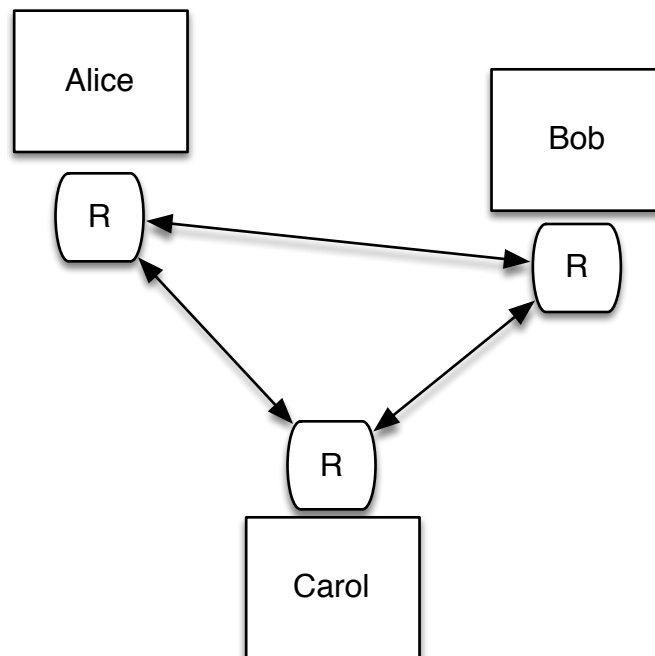
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of "It's really pretty simple, just think of branches as..."; and eventually you'll learn the commands that will fix everything.



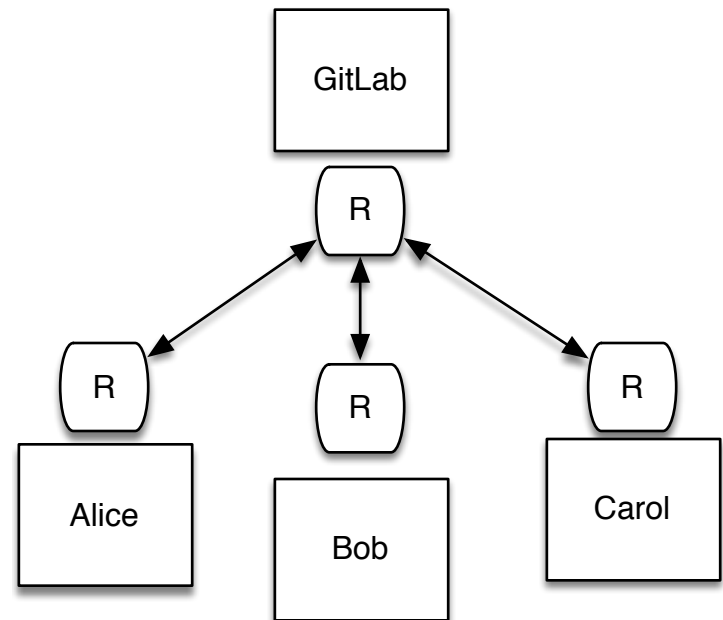
Git basics – general version

- A project lives in a repository
- Each user has their own copy of the repository
- A user *commits* changes to her copy to save them
- Other users can pull changes from that repository



Git basics – central repo (we'll use)

- Users have a shared repository (called *origin* in the git literature; for cse374 it is your group's repository on the CSE GitLab server)
- Each user *clones* the repository
- Users *commit* changes to their local repository (clone)
- To share changes, *push* them to GitLab after verifying them locally
- Other users *pull* from Gitlab to get changes (instead of from each other)



Tasks

Learn the common cases; look up the uncommon ones.

In a production shop using git...

- Create
 - a new repository/project (rare – once or twice a year)
 - a new branch (days to weeks; not in cse374, but used in production shops for independent development)
 - a new commit (daily or more, each significant change)
- Push to repo
 - regularly, when you want to back up or share work – even with yourself on a different computer
- Other operations as needed (check version history, differences, ...)

Repository access

A repository can be:

- Local: run git commands in repo directory or subdirectory
- Remote: lots of remote protocols supported (ssh, https, ...) depending on repository configuration
 - Specify user-id and machine
 - Usually need git and ssh installed locally
 - Need authentication (use ssh key with GitLab)
- cse374/HW6 use ssh access to remote GitLab server
- Feel free to experiment with private, local repos or private repos on gitlab

Getting started (GitLab)

- Create local ssh keys (ssh-keygen) and add to your GitLab account (instructions on gitlab, linked from cse374 git tutorial; only need to do this once)
- Set up a repository (we'll do this for you on hw6; if you do it yourself you get to pick name, location)
 - +New Project (on gitlab dashboard)
- Clone a working copy of the repo to your machine
 - cd where-you-want-to-put-it
 - git clone git@gitlab.cs.washington.edu:path/to/repo
 - url for above comes from gitlab page for your project, find using link in email you get when project created or on by logging in to gitlab
 - If git asks for password, keys aren't set up right – fix it

Routine git/GitLab local use

- Edit a file, say `stuff.c`
- Add file(s) to list to be saved in repo on next commit
`git add stuff.c`
- Commit all added changes
`git commit -m "reason/summary for commit"`
- Repeat locally until you want to push accumulated commits to GitLab server to share with partner or for backup...

git/GitLab use (sharing changes)

- Good practice – grab any changes on server not yet in local repo
 git pull
 – Also do this any time you want to merge changes pushed by your partner
- Test, make any needed changes, do git add / git commit to get everything cleaned up locally
- When ready, push accumulated changes to server
 git push
- If push blocks because there are newer changes on server, do a git pull, accept any merge messages, cleanup, add/commit/push again

File rename/move/delete

- Once files have been committed to gitlab repository, need to tell git about any changes desired to git-managed files

`git mv files`

`git rm files`

- git will make the changes locally then make corresponding changes to remote GitLab repo when you push
- If you use regular shell mv/rm commands, git will give you all sorts of interesting messages when you run git status and you will have to clean up 😊

Demo



Some examples

- Update local copy to match GitLab copy
git pull
- Make changes
git add file.c
git mv oldfile.c newfile.c
git rm obsolete.c
- Commit changes to local repo
git commit -m "fixed bug in getmem"
- Examine changes
git status (see uncommitted changed files, will also show you how to revert changes, etc.)
git diff *file* (see uncommitted changes in *file*)
git log (see history of commits)
- Update GitLab shared repo to reflect local changes
git push

Conflicts

- This all works great if there is one working copy.
- But if two users make changes to their own local copies, the two versions must be *merged*
 - git will merge automatically when you do a “git pull”
 - Usually successful if different lines or different files changed
- If git can’t automatically merge, you need to fix manually
 - git will tell you which files have conflicts (git status)
 - Look in files, you will see things like

```
<<<<<<<< HEAD
for (int i=0; i<10; i++)
=====
for (int i=0; i<=10; i++)
>>>>>>> master
```
 - Change these lines to what you actually want, then add/commit the changes (and push if you want to)

git gotchas

- Do not forget to add/commit/push files or your group members will be very unhappy
- Keep in the repository *exactly* (and *only*) what you need to build the program
 - Yes: foo.c foo.h Makefile
 - No: foo.o a.out foo.c~
 - You don't want versions of .o files etc.:
 - Replaceable things have no value
 - They change a lot when .c files change a little
 - Developers on other machines can't use them
- A simple .gitignore file can be used to tell git which sorts of files should not be tracked (*.o, *~, .DS_Store (OS X))
 - Goes in top-level repo directory; useful to push to GitLab and share

Summary

- Another tool for letting the computer do what it's good at:
 - Much better than manually emailing files, adding dates to filenames, etc.
 - Managing versions, storing the differences
 - Keeping source-code safe
 - Preventing concurrent access, detecting conflicts
- git/Gitlab tutorial for CSE 374 on website
- Links to GitLab on website and in CSE 374 tutorial
 - And in the mail you get when your repo is created
- Full git docs and book are online, free, downloadable
 - Beware of complexity – much of what they describe is beyond what we need for CSE 374; keep it simple