CSE 374 Lecture GIT

Version control and Git

What is version control?

Subversion, perforce, mercurial, cvs, sourcesafe, **git**

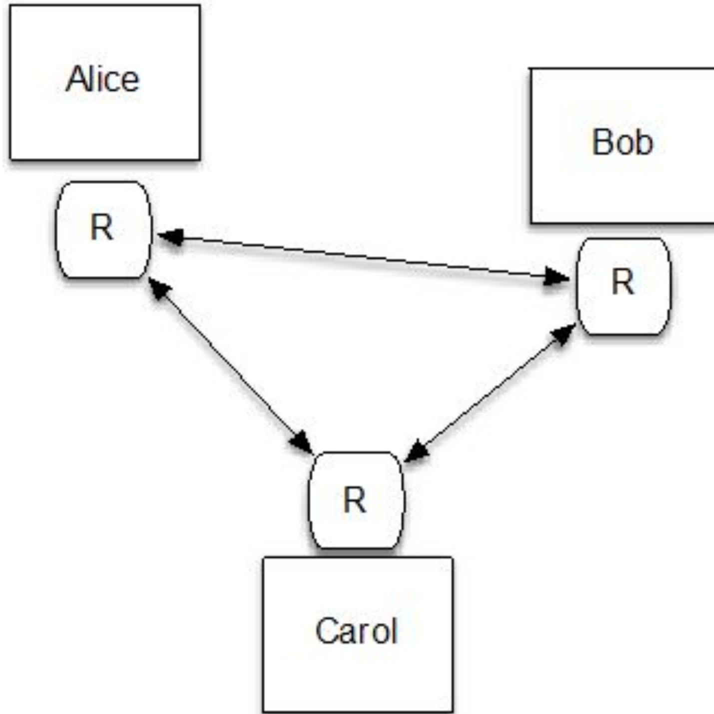
Software system that keeps records of files, changes-to-files, and manages sharing them between collaborators.

Why is version control?

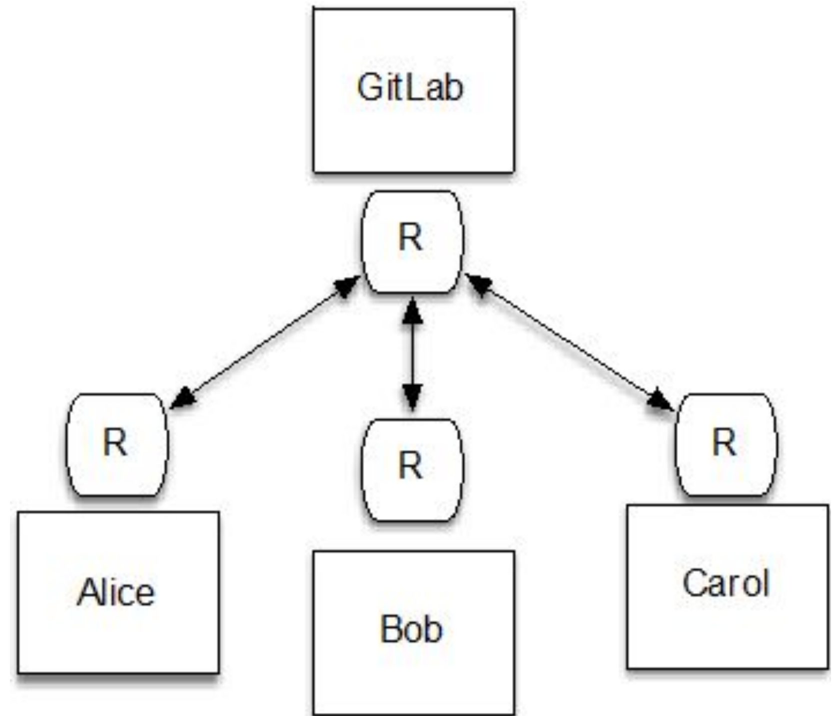
1. Backups. Archives a project to keep a safe copy.
2. Collaboration. Keeps a shared copy of project that all collaborators can access and update. Manages concurrent and maybe conflicting changes.
3. Version log. Keeps copies of previous versions so collaborators can revert if necessary.

Notes: Not language or coding specific; version control is used for all types of documents.

Alternate Models



Distributed System



Centralized System

General Repository Model

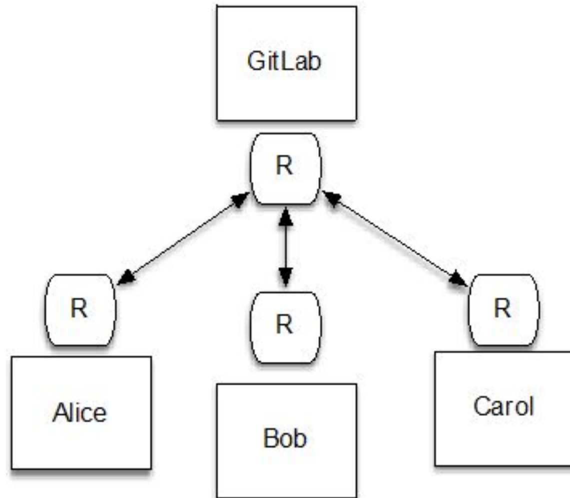
- A project lives in a collection called a "repository" (essentially a folder).
- Each user has their own copy of the repository (in the following diagram notated with "R").
- A user "commits" changes to their copy of the repository to save them.
- Other users can "pull" changes from that repository into their own local repository.

Decentralized: Data is stored in local repositories.

Users can fork from other user's repositories.

Does not scale well to large numbers of users.

Git Centralized Repository



- Users have a shared repository (“origin” or “remote”) which lives on a central server.
- Each user "clones" the repository to create a "local" copy.
- A user "commits" changes to their copy to save them.
- To share changes, a user "pushes" their local changes to the origin.
- All users "pull" from the central server periodically to get changes (instead of from each other).
- We call the central repository the "remote" repository; access to the remote repository requires authorization.
- Once code is pushed to the central server, the history of commits is linear. Commits may be reverted.

What is Git?

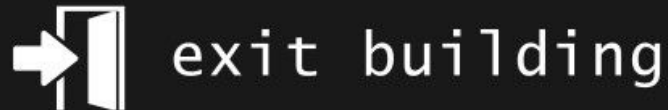
If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of "It's really pretty simple, just think of branches as…"; and eventually you'll learn the commands that will fix everything.

What do we do with Git?

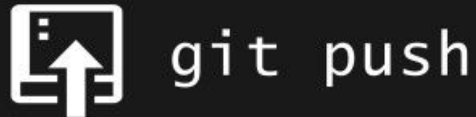
Learn the common cases; look up the uncommon ones.

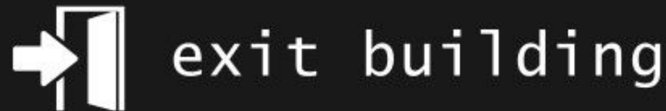
- Create
  - a new repository/project (rare – once or twice a year)
  - a new branch (days to weeks; not in cse374, but used in production shops for independent development)
  - a new commit (daily or more, each significant change)
- Push to repo
  - regularly, when you want to back-up or share work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

IN CASE OF FIRE



IN CASE OF FIRE





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 use ssh access to remote GitLab server
- Feel free to experiment with GitLab

Getting Started

- 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 have a few to start with; 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

Inbox (3) - m | Files · main | Megan Haz | CSE 374 - 2 | Grading Das

gitlab.cs.washington.edu/mh75

W Menu

Megan Hazen
@mh75

Set status

Edit profile

Preferences

Sign out

W Menu Search GitLab

User Settings

- Profile
- Account
- Applications
- Chat
- Access Tokens
- Emails
- Password
- Notifications
- SSH Keys
- GPG Keys
- Preferences
- Active Sessions
- Authentication log

User Settings › SSH Keys

Search settings

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

To add an SSH key you need to generate one or use an existing key.

Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/ '~/.ssh/id_rsa.pub' and begins with 'ssh-ed25519' or 'ssh-rsa'. Do not p that can compromise your identity.

Typically starts with "ssh-ed25519 ..." or "ssh-rsa ..."

Title e.g. My MacBook key

Expires at mm/dd/yyyy

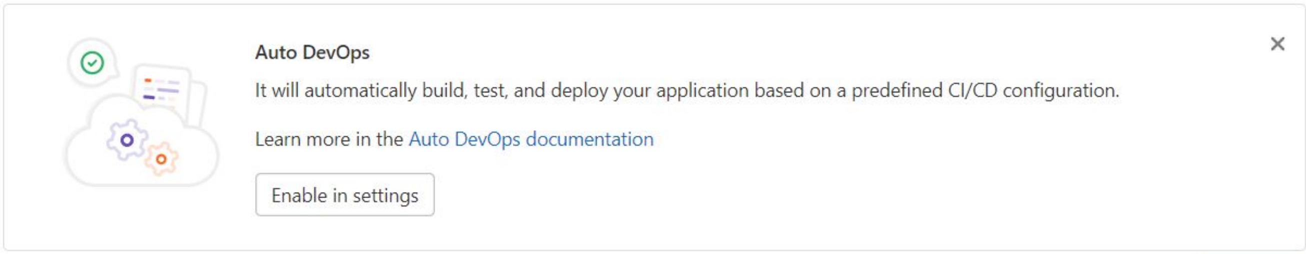
```
[mh75@localhost ~]$ git clone git@gitlab.cs.washington.edu:mh75/cse374-22wi.git
Cloning into 'cse374-22wi'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 2), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), done.
Resolving deltas: 100% (2/2), done.
[mh75@localhost ~]$ git pull
fatal: not a git repository (or any parent up to mount point /)
```

CSE374-22WI

Project ID: 73031

3 Commits | 1 Branch | 0 Tags | 113 KB Files | 113

Shared project for the 22 WI quarter



Clone with SSH

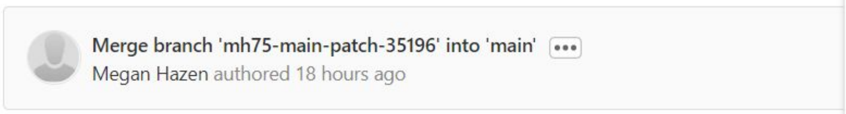
`git@gitlab.cs.washington.edu:mh75/`

Clone with HTTPS

`https://gitlab.cs.washington.edu/m`

Open in your IDE

Visual Studio Code (SSH)



Name | Last commit

Local Additions & Editing

- Edit a file "`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 commit -m 'messages should be useful'

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

File *mv* or *rename*

- Once files have been committed to gitlab repository:

  `git mv` *`files`*

  `git rm` *`files`*

  - git will make changes locally then update the 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

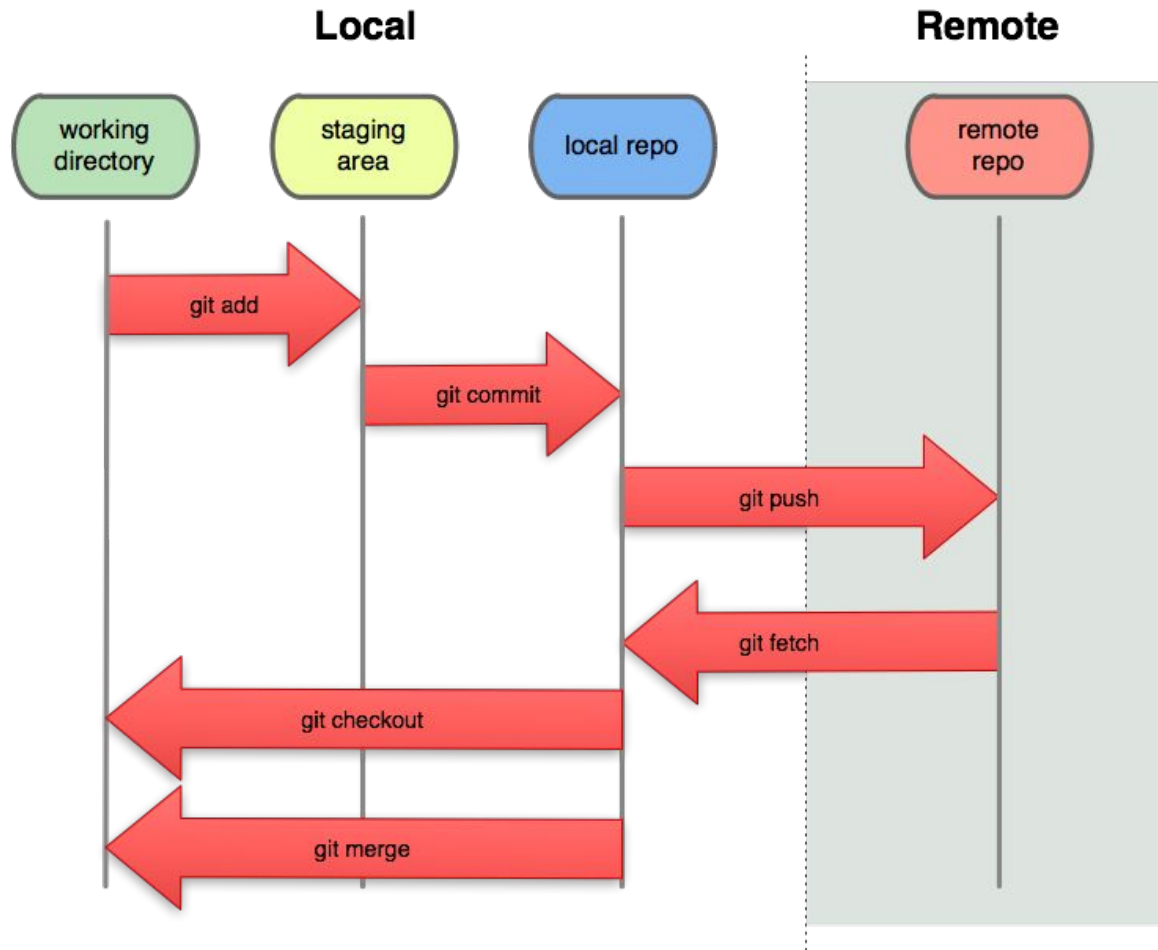
Gitlab remote use: sharing changes

- Good practice – update with remote changes:

  `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


Local
Remote
working directory
staging area
local repo
remote repo
git add
git commit
git push
git fetch
git checkout
git merge

Resolving Conflicts

git will tell you which files had merge conflicts (use `git status` to see conflicts), and the files will be edited to identify the conflict:

```
<<<<<<<< HEAD
for (int i=0; i<10; i++)
===============
for (int i=0; i<=10; i++)
>>>>>>>> master
```

You must modify the section to contain the code you want, then save, add, and commit the merge.

When more than one person works on a file you may get ‘merge conflicts’.

A merge conflict occurs when a person ‘git pull’s a file that has been updated on the remote repository since they started editing.

If git detects a merge issue (the same line of code edited in two non-sequential commits, i.e. commits made at the same time), it will attempt to automatically resolve the merge.

But if the commits modified the same portion of a file you will have to fix the conflict manually.

Example Commands

- Update local copy to remote

```
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 segfaut in getmem"
```

Examine changes

`git status` (see uncommitted changed files, or 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
```

Fixing Mistakes

- Set local repository to the last commit (forget all changes that you've made), you can run `git reset --hard HEAD`
- Here "HEAD" refers to the most recent commit.

- If one of your past commits was BAD, you can undo it using
`git revert`
- If the second-to-last commit was bad, you can undo it by saying
`git revert HEAD~1`
   a. HEAD is the most recent commit and "1" signifies the one before it. This will create a NEW commit that is the opposite of the original commit.
- Commits aren't completely static and permanent. If you make a commit but then realize you forgot one little thing, you can "amend"/modify your previous commit
`git commit --amend`

.gitignore

Git may be used to store any types of files.

HOWEVER

Do not store files that are unnecessary.

- ➔ Backup files (like *~ emacs backups)
- ➔ Files that can be recreated (such as .o files) should not be added.
- ➔ System specific files

'.gitignore' lists files not to upload to HEAD

```
# emacs backup files
*~

# OS X finder info files
.DS_Store

# built object files
*.o
```

374: Gitlab

Resources on line -

https://gitlab.cs.washington.edu/help

https://courses.cs.washington.edu/courses/cse374/22wi/resources/git.html

https://git-scm.com/book/en/v2

https://about.gitlab.com/images/press/git-cheat-sheet.pdf

You may choose to work with a partner for HW6, in which case, you must use a CSE GitLab repository to store all of the code and other files associated with the project.

Don't store things like .o files and executable programs that don't belong in a repository.

You must use the provided repository even if you have separate machines or accounts of your own that you use for other projects.

Both partners should be regularly committing and pushing changes to your repository. The git log to reflect reasonable activity by both members of the group.