Version Control

C Programming and Software Tools

N.C. State Department of Computer Science

Computer Science
NC STATE UNIVERSITY

Version Control Systems

- Functions
  - convenient, secure access by many people to a shared project
  - easy to backup to a remote server
  - flexible reversion to a previous version
  - conflict control between multiple developers of a project
- What are some version control systems?
  - **`subversion`** – centralized
  - **`Git`** and **`GitHub`** – decentralized

What is Git?

- Decentralized version control
  - Developed by Linus Torvalds
  - Help with Linux development
  - Easier to manage volunteer code contributions
- A codebase is in a repository
  - Not a client-server model like SVN
- Code is moved between repositories by pulling and pushing
- All repos are created equal

What is Git*Hub*?

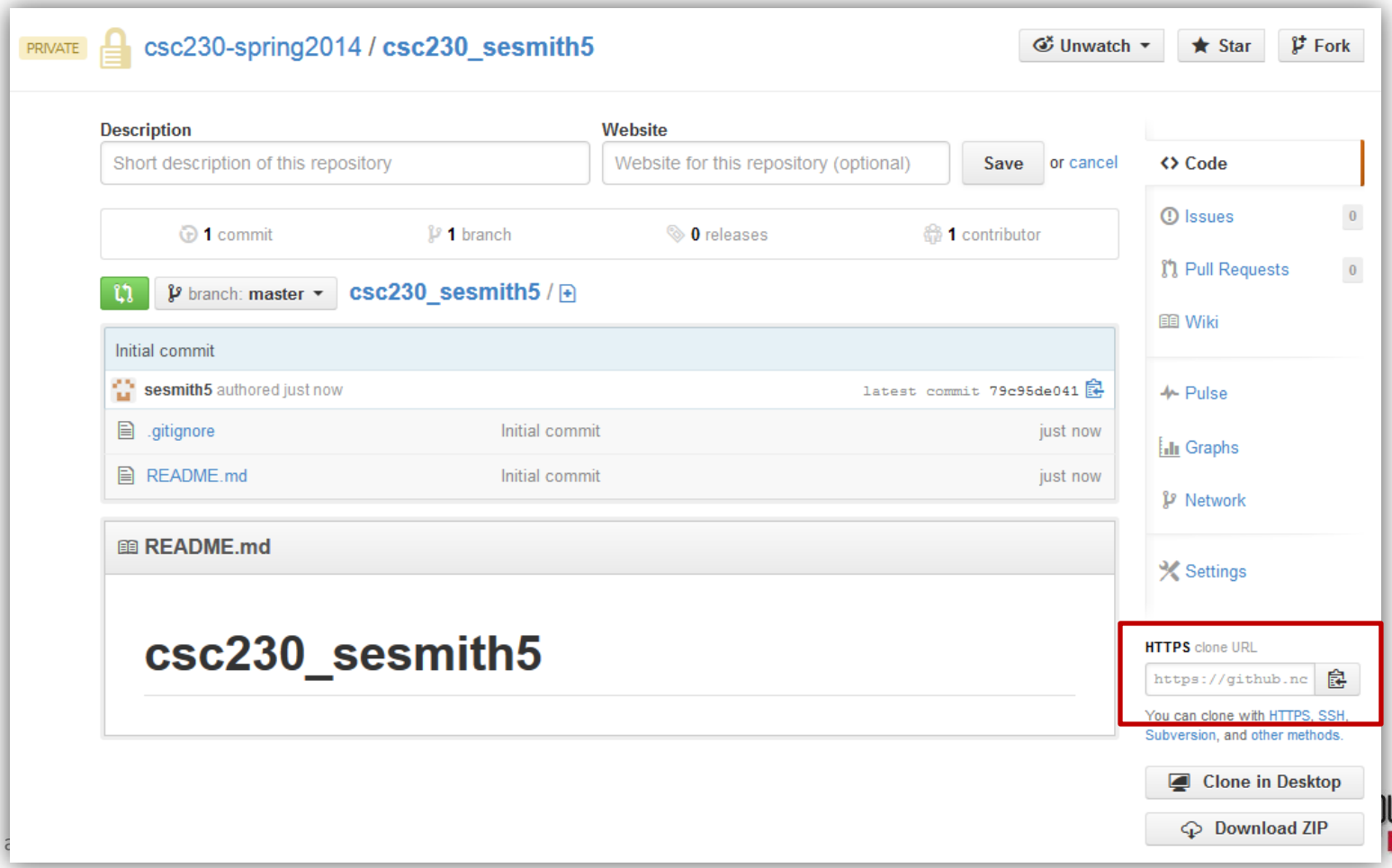
- GitHub is a service that hosts Git repos in the cloud
  - Just like a local repo
- Additional features:
  - Wikis
  - Bug tracking

Working with GitHub

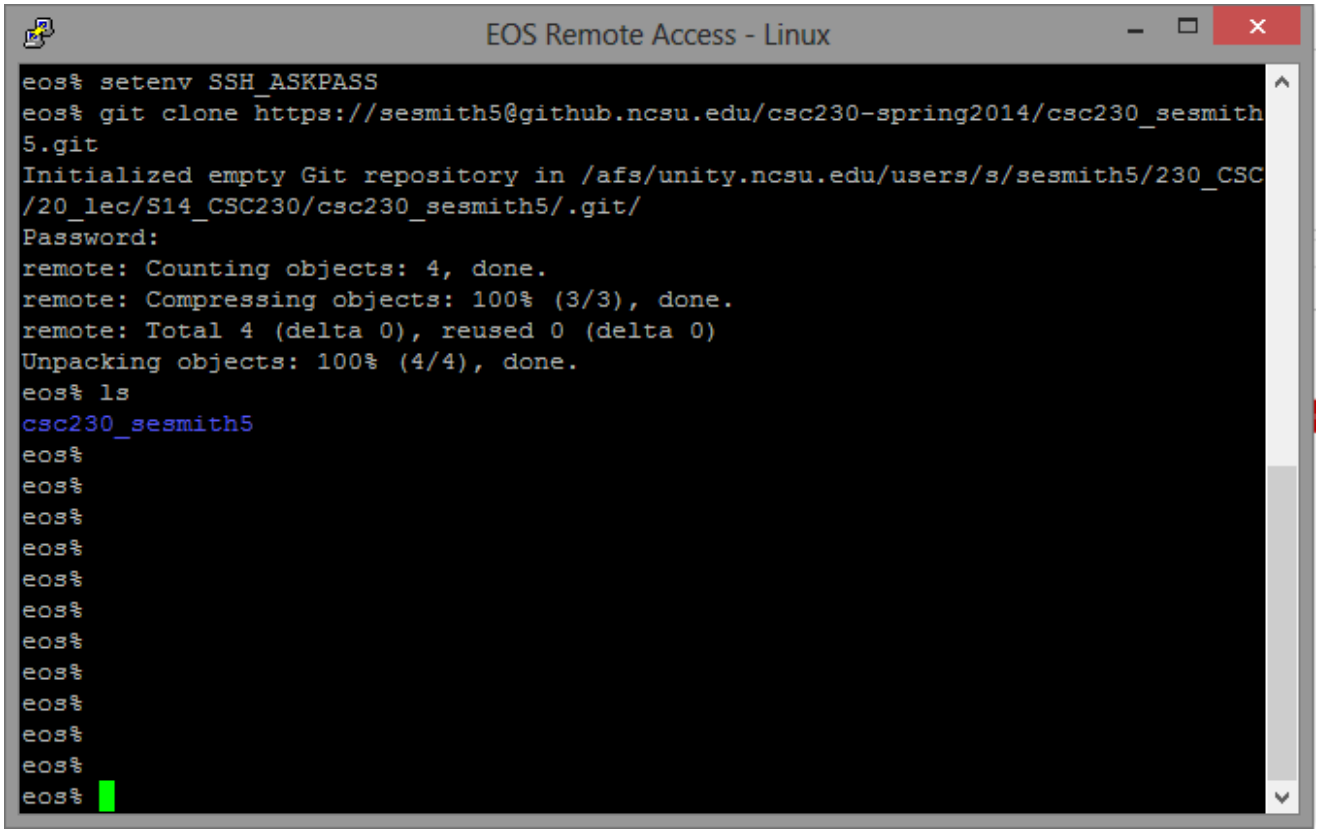
- We will be using github.ncsu.edu
  - Same ideas can be used on github.com
- We will be providing repos that MUST be used for CSC230
- You can create your own repos for other projects and classes
  - Makes sure that repo names couldn't be possible unity ids!
  - Course works MUST be in private repos

Start Working…

- Go to github.ncsu.edu and copy the HTTPS clone URL



Clone the Repo




```
$ unset SSH_ASKPASS
```

Turn off really stupid graphical password prompt that NCSU defaults to for some reason

```
$ git clone https://<unity_id>@github.ncsu.edu/<org_name>/<repo_name>.git
```

Git Workflow

- Create, edit, delete your files like a normal
- If you have created new files locally that should be added to the repository, run

```
$ git add .
```

  - The add command will associate all new files with the repo

- Commit the files to your repo

```
$ git commit –am "A meaningful commit message"
```

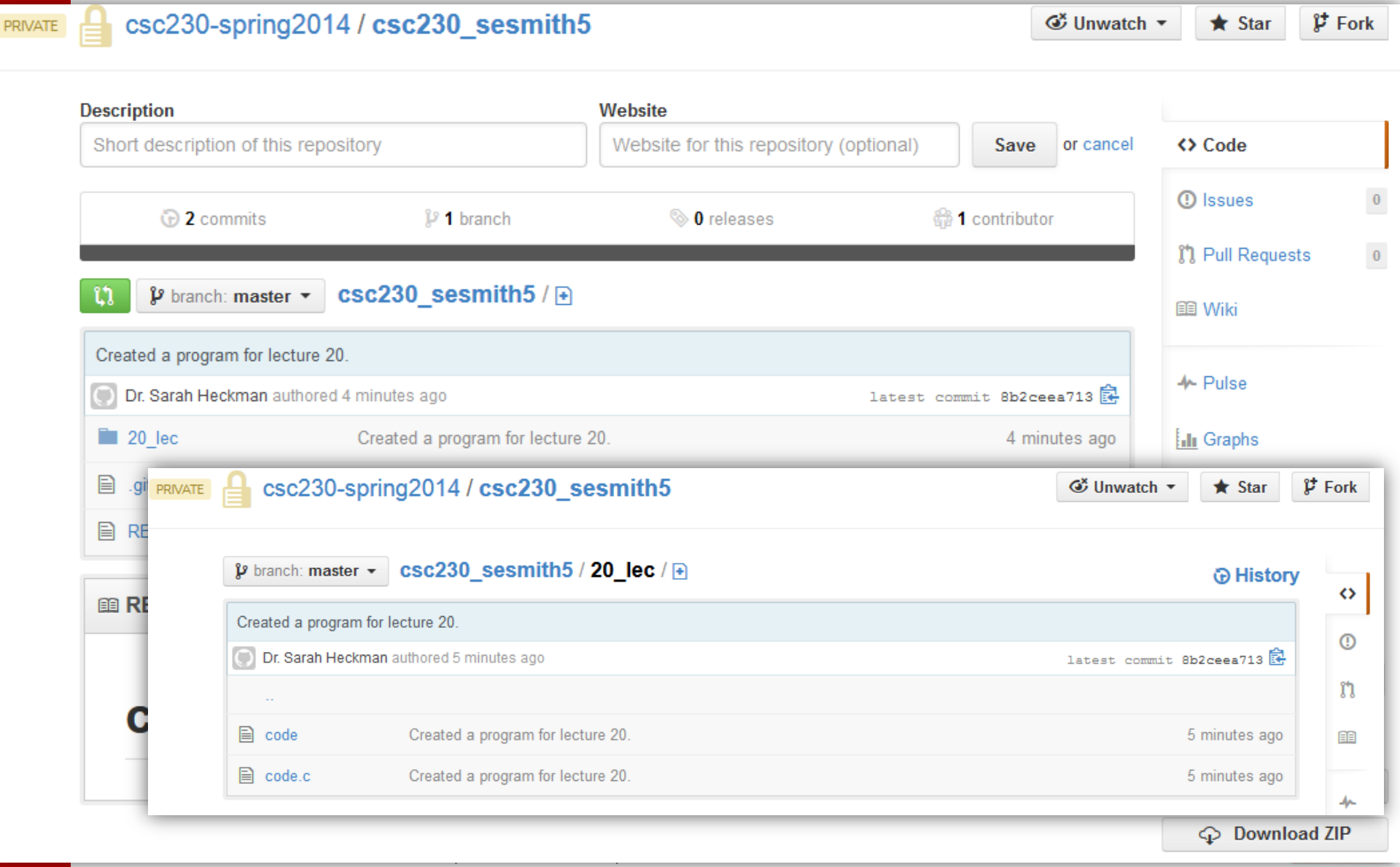
  - Only commits the code to your local repo!

Git Workflow – Between Repos

- Code is *pushed* from a local repo to a remote repo
  - If you want to move your local changes to GitHub, you must push your code
  - This pushes the code to the repo that you cloned
  - Will be used for homework submission

```
$ git push
```

Verify Your Changes!



Git Workflow – Between Repos

- Code is *pulled* from a remote repo to a local repo
  - If you are working on multiple machines, you can use the remote repo as an archive
  - If you are collaborating with multiple people, the remote repo becomes the collaborative space
  - Always pull the latest from the remote repo before making new changes

```
$ git pull
```

Common Mistakes

- Committing without pushing
  - You have two repos – a local repo AND a remote one
  - *Commit* to the local repo
  - *Push* to the remote repo (GitHub)
- Always check your repo on the website to make sure the files you want are there!

Organizing Your Repo

- All future homework submissions will be though NCSU's GitHub and your provided GitHub repo
- We use Jenkins to automatically pull, compile, and test your programs!
  - You will have an estimate of a portion of your homework grade BEFORE the deadline!
- You will create a folder for each homework
  - We will provide the naming conventions so that your repo will work with Jenkins

Preparing for HW3

- As part of HW2, you are expected to log into GitHub
  - It's 10 VERY easy points!
  - Actually go log in now!
- If we are unable to search for you in GitHub, you won't earn the 10 points on HW3 and you won't be ready for future homeworks

BACKUP

SVN Clients

- There are many SVN clients
  - Command line
    - SVN
  - GUI
    - SmartSVN
  - IDE plug-ins
    - Eclipse: Subversive
    - Visual Studio: VisualSVN
  - File system integrations
    - Windows: TortoiseSVN, Smart SVN

See: http://en.wikipedia.org/wiki/Comparison_of_Subversion_clients

Repositories

- A repository is where the backup (master) copies of all files are kept
- Imaginary server for this discussion: **`subversion.ncsu.edu`**
- Must have user account on server to create a repository, using the command **`svnadmin`**
- Clients maintain local (working, individual) copies of files on the server

Starting a Project

- Create a temporary directory (e.g., **`/home/bill/myproj`**), with subdirectories…
  - **`branches`** (initially empty)
  - **`trunk`** (has files you want to be part of the project)
  - **`tags`** (names of releases or milestones)
- Import these files into the svn repository

```
% svn import /home/bill/myproj ↵
    svn://subversion.ncsu.edu/repos1
Adding     /home/bill/myproj/trunk
…more svn output…
Committed revision 1.
```

Checking Out: Example

- Importing project files into the repo does NOT connect the local directory to the repo in any way
- Need to check out project
- Check out latest revision from the repository

```
% svn co
  svn://subversion.ncsu.edu/repos1/proj1
A  proj1/trunk
A  proj1/trunk/search.c
…
Checked out revision 4.
```

Revisions

- A *revision* is a snapshot of project at one moment in time
  - allows users to say "get revision 1432 of XYZ"
- **`commit`** creates a new revision

Committing a Revision: Example

- Update to latest revision from the repository

```
% svn update
A  proj1/trunk
A  proj1/trunk/search.c
…
Checked out revision 4.
```

Edit **search.c**

```
% cd proj1 ; vi trunk/search.c
```

Commit the changes to the repository (new revision)

```
% svn commit –m “Add better search”
Sending trunk/search.c
Transmitting data…
Committed revision 5.
```

Getting File Info

- Get info on a particular file or directory

```
% svn info trunk/search.c
Path: trunk/search.c
Url:
  svn://subversion.ncsu.edu/repos1/trunk/search.c
Revision: 5
Node Kind: file
Schedule: normal
Last Changed Rev: 5
Last Changed Date: 2006-08-08 12:20:18 -0700
  (Thu, 08 Aug 2006)
```

The Basic Steps in Using svn

1. Check out a working copy
2. Update your working copy (modify, add, delete files and folders)
3. Make changes to selected files
4. Examine your changes
5. Merge with other people's changes
6. Commit your changes

Basic Steps

- 1. Check out a working copy

```
% svn co svn://subversion.ncsu.edu/repos1 ↵
  proj1
```

- 2. Update the working copy
  - Update all files and directories to the most current version

```
% svn update
```

  - Get an older revision for all files

```
% svn update -r 1345
```

  - Get an even older version of a particular file

```
% svn update -r 999 search.c
```

…Basic (cont'd)

- 3. Make changes to local copies of files
  - Add new files and directories

```
% vi trunk/new_algorithm.c
% mkdir trunk/data-files
% vi trunk/data-files/file1
% svn add trunk/new_algorithm.c ↵
trunk/data-files
```

  - Delete files

```
% svn delete foo old_algorithm.c
```

…Basic (cont'd)

- Rename file

```
% svn rename trunk/README.txt ↵
trunk/README_OLD.txt
```

- Copy files and directories

```
% svn copy trunk/data-files ↵
trunk/data-files-new
```

…Basic (cont'd)

- 4. Examine your changes (more info with –v)
- `% svn status`

```
_ L     ./abc.c
```
[svn has a lock in its .svn directory for abc.c]

```
M       ./bar.c
```
[the contents in bar.c have local modifications]

```
?       ./foo.o
```
[svn doesn't manage foo.o]

```
!       ./foo.c
```
[svn knows foo.c but a non-svn program deleted it]

```
A  +    ./moved_dir
```
[added with history of where it came from]

```
M  +    ./moved_dir/README
```
[added with history and has local modifications]

```
D       ./stuff/fish.c
```
[this file is scheduled for deletion]

…Basic (cont'd)

- **`svn status -v`** (be verbose)
  - second column, working revision
  - third column, last changed revision
  - fourth column, who changed it

```
% svn status -v
M      44    23    joe      ./README
       44    30    frank    ./INSTALL
M      44    20    frank    ./bar.c
       44    18    joe      ./stuff
       44    35    mary     ./stuff/trout.c
D      44    19    frank    ./stuff/fish.c
       44    21    mary     ./stuff/things
A       0     ?     ?       ./stuff/things/bloo.h
       44    36    joe      ./stuff/things/gloo.c
```

…Basic (cont'd)

- **`svn diff`**: Show your modifications
  - show all differences between files in repository (most recent revision) and local working copy

```
% svn diff
```

  - diff between revision 3 of **`foo.c`** in repository and local working **`foo.c`**

```
% svn diff -r 3 foo.c
```

  - diff between revisions 2 and 3 of **`foo.c`** in the repository

```
% svn diff -r 2:3 foo.c
```

…Basic (cont'd)

- Revert (i.e., discard your changes)
  - (does not require network access)

```
% svn revert . -R
```

- 6. Commit your changes (create a new revision)

```
% svn commit
```

Conflict Resolution

- Conflicts occur when two users are working independently on their own local copies of the same file (e.g., **`pgm.c`**)
  - first and second users update the file: revision 4
  - first user commits their changes (revision 5): no problem
  - second user commits their changes: conflict!
  - indicated by a **`C`** in **`svn update`** output
- The resulting "committed" file **`pgm.c`** has embedded conflict markers

Conflicts... (cont'd)

- Three temporary files are also created
  - **`pgm.c.mine`** – 2nd user's (uncommitted) file
  - **`pgm.c.r4`** – file 2nd user checked out, before any changes committed by either user
  - **`pgm.c.r5`** – file containing 1st user's changes, without 2nd user's changes
- Subversion requires definite action from the user 2 to resolve the conflict

Conflicts... (cont'd)

- Possible resolutions
  1. hand merge the conflicting text in **`pgm.c`**, or
  2. copy one of the temporary file versions over **`pgm.c`**, or
  3. run **`svn revert`** to undo all of your changes
- 5. Once resolved, you need to tell svn that the conflict has been resolved

```
% svn resolved pgm.c
```

  – also deletes the temporary files

Locks

- *locking* = a mechanism for mutual exclusion between users to avoid clashing commits
- Creating a lock

```
svn lock trunk/search.c -m ↵
   "On a deadline, pls do not modify"
```

- Succeeds if the file isn't already locked by someone else, and is up to date
- Attempts by other users to update the master version of the file (through `svn commit`) will fail, with an error message

Locks (cont'd)

- Releasing a lock:
  - `svn unlock <filename>`, or
  - `svn commit` (automatically releases locks)

Branches

- *Branches* are parallel copies of projects, maintained by subversion
- Often: one main (production) branch, and many versions (in development branches)
- Can be edited and modified separately, but share common files
- Can be merged when a branch has been fully tested

Branches (cont'd)

- Creating a branch
  - `svn copy trunk branches/br1`
  - `svn commit -m "Created branch br1"`
- Checking out just a branch
  - `svn checkout ↵`
    `svn://subversion.ncsu.edu/repos1/branches/br1`
- Merging two branches
  - `svn merge`
    `svn://subversion.ncsu.edu/repos1/trunk ↵`
    `svn://subversion.ncsu.edu/repos1/branches/br1 trunk`