C S 80A: OPEN SOURCE CONTRIBUTION

Foothill College Course Outline of Record

Heading	Value
Effective Term:	Summer 2021
Units:	4.5
Hours:	4 lecture, 2 laboratory per week (72 total per quarter)
Advisory:	C S 40A; C S 1A and 1B (or 1M), or 2A and 2B.
Degree & Credit Status:	Degree-Applicable Credit Course
Foothill GE:	Non-GE
Transferable:	CSU
Grade Type:	Letter Grade (Request for Pass/No Pass)
Repeatability:	Not Repeatable

Student Learning Outcomes

- A successful student will be able to join a team that handles the workflow of a specific open-source project and become a productive contributor to such a team.
- A successful student will be able to install a Git repository and issue the various commands for checking-in, checking-out, and forking a project's source code.

Description

Introduction to the tools for, and culture of, contributing to open source software projects. Tool-based topics include Git repositories, pull requests, forks, logs, merges, tagging, rebasing and server configuration. Concept topics include commit guidelines, branching workflows, small-team vs. large-team workflows, project maintenance, iterative staging, selecting viable source communities, joining public projects, setting up accurate dev environments, testing and prepping patch merges, and becoming a committer.

Course Objectives

The student will be able to:
A. Install a Git client repository (repo).
B. Perform the basic Git commands of repository check-in and check-out.
C. Use the Git log to reveal differences in version snapshots.
D. Pull-from and push-to remote repos.
E. Tag commits for version tracking.
F. Perform the all the commonly used branching commands.
G. Explain different styles of branching workflows.
H. Set up a Git server.
I. Describe the main aspects of collaborative project participation.
J. Demonstrate a mastery of Git project maintenance tools.
K. Use Git tools such as stashing and iterative staging.
L. Select a viable open source project for contribution.
M. Research the project culture and duplicate the development environment locally.
N. Identify a problem or bug that needs fixing.
O. Explain how to become a project committer.
P. Establish a term project and demonstrate successful milestones leading toward an open source contribution.

Course Content

A. Setting Up Git Tools
1. Installing Git (client)
2. Configuring Git
3. Establishing a GitHub account
B. Git Basics
1. Stack structure of Git
2. Creating repos with init vs. clone
3. File status lifecycle
4. Commands: status, remove and reset
5. Ignoring files
C. The Git Log
1. Using log
2. Using diff
3. Hash tags and the SHA-1 protocol
D. Remote Server Operations
1. Clone
2. Add
3. Fetch
4. Push/pull
5. Removing and renaming
E. Tagging
1. Annotating tags
2. Signed tags
3. Lightweight tags
4. Verifying tags
F. Git Branches
1. Tree structure of branches
2. Forking (creating branches)
3. Checking-out branches
4. Graph structure of merged branches
5. Merging
G. Branching Workflows
1. Forking a development branch
2. Forking a topic branch
3. Forking a hotfix branch
4. Tracking and deleting remote branches
5. Destroying remote branches
6. Rebasing and executing fast-forward
H. Setting up a Git Server
1. Protocols (local, Git, HTTPS, SSH)
2. Public keys and Gitosis
3. Configuring a server
4. Web-based visualizers
5. Hosted Git: GitHub
I. Project Contribution with Git
1. Commit guidelines
2. Private: small-team vs. large-team
3. Public: small-team vs. large-team
J. Project Maintenance
1. Topic branch options
2. Checking-out remote branches by manager
3. Examining contributed work
4. Integrating contributed work
5. Tagging releases
6. Build numbers
7. Shortlog
K. Special Git Techniques

1. Stashing
2. Interactive staging
3. Short SHA
4. Changing history
5. Debugging
6. Submodules

L. Viable Open Source Projects
1. Selecting an open source that is of interest
2. Reading the public discussion forums
3. Evaluating the code through user comments
4. Evaluating the code through contributor response
5. Reading the code docs

M. Getting on Board
1. Watching projects
2. Pull requests
3. Setting up a dev environment
4. Unbundling tests for student environment

N. Fixing Bugs and Solving Problems
1. Solving a problem that the student needs solved
2. Confirming the problem is the code's, not student's
3. Confirming the problem is supported by the forums
4. Making student patch mergeable
5. Understanding the test requirements for the student patch
6. Asking the forums about student patch idea
7. Submitting student patch

O. Becoming a Committer
1. Getting commit privileges
2. Writing good documentation
3. Contributing regularly
4. Making relevant commits

P. Project Plan Timeline
1. Instructor list of suggested repos
2. Student repo proposal and forum participation
3. Acceptance of proposal by instructor
4. Student demonstration of working dev environment
5. Student demonstration of successful patch
6. Student demonstration of documentation of patch
7. Student demonstration of patch pushed to community

Lab Content

A. Installation of Client Git Environment
1. Download and install a Git environment
2. Create a local repository using clone and init
3. Modify and commit changes
4. Use log to track changes

B. Establishing Remote Repositories
1. Create a local repository (repo) on GitHub
2. Use the local computer to connect to the remote repo
3. Do an initial push to the remote repo
4. Do a pull from the remote repo

C. Creating Tagged Branches
1. Commit annotated branches to your repo
2. Use lightweight tags in your project
3. Verify the tags using Git's verification support
4. Create a hotfix or new topic branch and establish parallel development separate from the master branch
5. Check-in the new branch and merge the new branch with the master

D. Server and Project Local Contribution Exercise
1. Set up a Git server on a local computer
2. Select a protocol for transfer and configure the server to use it
3. Establish public access and confirm that other computers can reach and connect to the server
4. Establish commit guidelines for projects for this server
5. Demonstrate repo check-ins to this server that involve tagged releases, interactive staging and/or history changes

E. Final Open Source Project Phase 1
1. Use your local lab system to research a viable open source project
2. Interact with the current contributors in public forums and document this discussion
3. Report on existing public commentary on the code and current bugs
4. Summarize the code docs you find on the project site

F. Final Open Source Project Phase 2
1. Issue a pull request and set up a development clone on your lab system
2. Test the repo to make sure it is an accurate representation of the public repo
3. Select a small bug on which you intend to work
4. Give supporting evidence that the forum contributors believe this bug to be worth fixing

G. Open Source Project Phase 3
1. Fix the bug
2. Test the fix using all available tests, both of your own design and those available from the community
3. Document the bug fix according to the standards of the open source community from which it comes

H. Open Source Project Phase 4
1. Report to the forums that you have a bug fix
2. Document the reaction and interaction of your report
3. Request that you be allowed to commit your branch to the public repo
4. Document the response and submit copies or screen shots of the public repo and/or the forums that indicate the bug has been received

Special Facilities and/or Equipment

A. The college will provide access to a computer laboratory with language compilers and internet connectivity.
B. The college will provide a website or course management system with an assignment posting component (through which all lab assignments are to be submitted) and a forum component (where students can discuss course material and receive help from the instructor). This applies to all sections, including on-campus (i.e., face-to-face) offerings.
C. When taught via Foothill Global Access on the Internet, the college will provide a fully functional and maintained course management system through which the instructor and students can interact.
D. When taught via Foothill Global Access on the Internet, students must have currently existing email accounts and ongoing access to computers with internet capabilities.

Method(s) of Evaluation

Tests and quizzes
Written laboratory assignments which include source code, sample runs and documentation
Final examination

Method(s) of Instruction

Lectures which include motivation for tools of Git, GitHub and Open Source
Online labs (for all sections, including those meeting face-to-face/on-campus), consisting of:
1. A programming assignment webpage located on a college-hosted course management system or other department-approved internet

environment. Here, the students will review the specification of each programming assignment and submit their completed lab work
2. A discussion webpage located on a college-hosted course management system or other department-approved internet environment. Here, students can request assistance from the instructor and interact publicly with other class members
Detailed review of programming assignments which includes model solutions and specific comments on the student submissions
In-person or online discussion which engages students and instructor in an ongoing dialog pertaining to all aspects of designing, implementing and analyzing programs
When course is taught fully online:
1. Instructor-authored lecture materials, handouts, syllabus, assignments, tests, and other relevant course material will be delivered through a college-hosted course management system or other department-approved internet environment
2. Additional instructional guidelines for this course are listed in the attached addendum of CS department online practices

Representative Text(s) and Other Materials

Fogul, Karl. Producing Open Source Software: How to Run a Successful Free Software Project. 2006.

Chacon, Scott. Pro Git. 2009.

Brasseur, VM. Forge your future with Open Source, 1st ed.. 2018.

The Fogul text is considered an industry classic.

Types and/or Examples of Required Reading, Writing, and Outside of Class Assignments

A. Reading
1. Textbook assigned reading averaging 30 pages per week.
2. Reading the supplied handouts and modules averaging 10 pages per week.
3. Reading online resources as directed by instructor though links pertinent to programming.
4. Reading library and reference material directed by instructor through course handouts.
B. Writing
1. Writing technical prose documentation that supports and describes the programs that are submitted for grades.

Discipline(s)

Computer Science