CS 543 — Adv SWE — SCM Project 3

SCM Project 3— Merged

Introduction

This is the third part of our SCM (Source Code Management) project. In this project part, we add the ability to merge two project trees (that are based on the same repo). Note that we already have a natural branching effect due to check-out (of a checked-in project version into a new project tree, AKA the Kid project version) coupled with tracking that project version's parent (AKA the Mom) version (as identified in the Kid's repo manifest).

The merge ability helps the user to merge a project tree (the R version) that is already in the repo (as represented by a manifest file) into a target project tree (the T version) outside the repo. We will assume that the T version has also just been checked-in, so that there is an up-to-date T version manifest file in the repo.

For example, Fred can merge Jack's changes (the R version, checked-in from Jack's project tree) that are in the repo into Fred's current project tree (the T version). If the merge succeeds (standard merge software is only able to handle simple file differences, but this could be the case with Fred and Jack) then Fred can quickly run some tests on the merged resulting project tree and check-in his resulting merged project tree for others to use. This is how a branch is merged back into mainline.

For project 3, we will only issue a command result that there are conflicting file versions and also leave copies of the conflicting file versions in the target project tree, so that Fred could look at the conflicting file versions and manually run his choice of merge software on the files that need it.

Team

The team may contain up to 3 members, and may be different from prior project teams. Pick a three-letter name for the team as described for project #2.

Merge Interface

The user interface for the merge operation is simple. (Note, one should always build a simple command line version of the UI so as to help maintain an MVC architecture, even if the end goal is to have a GUI.) The user needs to indicate the R project version by specifying its repo manifest file, for example by label or by filename. Also, the user needs to indicate the T project version. Because we can assume that the user has just checked-in the T project version, this T version can be indicated by the repo manifest file (name or label), or by merely giving the T version folder path – whichever is easiest. (In real life, we'd probably check in the T version from the folder path just to make sure.) Note, be sure to allow the R manifest to be selected by a label.

Merge Common Ancestor

The two project versions, R and T, to be merged were children of some prior Mom versions – R's Mom and T's Mom. Note that it's possible that their relationship is weird: R is T's Mom, or R is actually the repo root version via the Create command, or R is a deep ancestor of T – but we will assume this is not the case (because if it is, the merge results in no changes to T). Thus, w.l.o.g. we can assume that R and T are on different paths from the repo root. Then there is a common path from the root that eventually diverges, one path leading to R and the other path leading to T. And there must be a deepest common ancestor (AKA G for Grandma) for those two paths, that both paths share.

Project 3 must identify the G project version, because its files will be used if a 3-way merge is needed due to a given file being different (in conflict) in R and in T (something that can be determined by the file's two artifact ids: in R and in T.

CS 543 — Adv SWE — SCM Project 3

Merge Results

After the merge command has run, the target (T) project tree should have a single version (R or T, or both are the same) of all its conflict-free files, and also for each file where the R and T versions are different the target project tree should have three versions of that file each with an altered filename: an R version, a T version and a G version (obtained from the G project version). For example, if the conflicted filename is foo.java, then there should be in the target project tree in its place the three files: foo_MR.java, foo_MT.java, and foo_MG.java. (You can rename the target's foo.java to foo_MT.java)

The merge command should list (e.g., via standard output) the full pathname of at least the first conflicted file, or should indicate that the merge was a complete success. (The user can manually search for all the conflicted files, via for example the string "_MG", in the target project tree to determine exactly which specific files had a conflict and need special intervention.)

Testing

Be sure to provide cases that test for at least one conflict-file resulting in MT, MR, and MG files, and at least one non-conflict file (where the R and T files are identical, or where one of them doesn't exist).

Include, in your submitted .zip file, a sample "run" for each test, consisting of directory listings of the project tree and the repo, and of the new manifest file involved. These can be cut-and-pasted into a .txt file for the run.

Technical Debt

This is a class in S/W Engineering. In spite of that, we emphasize working S/W (Rule #0). However, getting to working S/W fast often leaves technical debt. Technical debt will rapidly turn into a Bad Smell if left too long to fester. Therefore, as this is the last rapid delivery, the technical debt must be paid, approximately in full: the source code should be reasonably clean and well-documented.

Academic Rules

Correctly and properly attribute all third party material and references, lest points be taken off.

Readme File

Same as for project #2.

Submission

Same as for project #2.

Grading

Same as for project #2.