CECS-543 — Adv SWEngr — VCS Project 3

VCS Project 3— Merge

Introduction

This is the third part of our VCS. In this project part, we add the ability to merge two project tree snapshots (that are based on the same repo, and hence the same development team project). Note that we already have a natural branching effect due to check-out (of a project snapshot, AKA the Kid) coupled with tracking that project version's parent snapshot (AKA the Mom, as identified in the Kid's repo manifest).

This merge ability is typically used to merge two snapshots that have different modifications (usually in different parts/files), or to merge a branch snapshot of modifications (e.g., a bug fix, or a new feature) back into a project mainline's snapshot.

We will do the merge in two parts: **merge-out** and **merge-in**. The **merge-out command** will gather the information needed for the user to manually do a 3-way merge. (Sorry, user, but this is a somewhat no-frills VCS.) The **merge-in command** will complete the merge by "checking in" a snapshot of the user's fully-merged project tree – the user did the full-merge by hand (again, no frills).

Also, note that our VCS is project-based, not file-based. We do not run a VCS command to copy an individual file to/from the repo, but instead only to copy an entire project tree (which resides outside the repo directory) to/from the repo (where “to repo” is taking a snapshot and “from repo” is recreating a new project tree from a snapshot).

Source & Target

The merge-out arguments are a repo source snapshot and a target project tree (actually that target's latest snapshot, from which we can extract the location of the target project tree). The source snapshot is the repo 'R' snapshot. The Target 'T' snapshot is assumed to be a snapshot that the user has just checked-in (sorry, user, you do it, no-frills). The Target project tree is in the user's project folder from which the T snapshot was (just) checked-in. The merge-out will (maybe) add new files/folders to the T project tree.

Merge File Collisions

These new files will include those in the Source that **don't match** (don't have the same artifact ID) as their corresponding target files – called **collision files**. Because we are merging an entire project tree, there may be many collision files for one merge-out command. When we detect that one of the many project snapshot files collides (has a different artifact ID) with the corresponding file of the other snapshot, we will add both files to the target project tree. The mismatching file from the repo source snapshot will be added, but with a suffix of “_MR” meaning “Mismatch-from-Repo”. The corresponding file from the target snapshot will have its filename suffixed with “_MT” meaning “Mismatch-in-Target”. Both files will retain their existing extensions (e.g, “fred_MR.java” and “fred_MT.java”). Also, a third corresponding file, the “_MG” file will be added to the target project tree; the “grandma” file.

Also, (optionally) the merge-out command will create a manifest file which includes the command and arguments, and the name of the “grandma” snapshot.

Grandma

In case of at least one file collision, you will need to find the “grandma” snapshot of the source and target snapshots. This is the **most recent common ancestor** snapshot. The “grandma” snapshot is the same for all pairs of colliding files. That Grandma 'G' snapshot is determined by the path of immediate

ancestor snapshots from a given (R or T) snapshot back to the root of the repo snapshot directed acyclic graph (DAG). The 'G' file will get its own suffix "_MG" meaning "Mismatch-from-Grandma" (e.g., "fred_MG.java"). Note that because of prior merges, there may be more than one path to the root from a snapshot; and then the grandma will be the most recent common ancestor of those common ancestors along any of those paths to the root.

Merge-In

The merge-in command assumes 1) that the user has finished manually fully-merging the R file changes into the corresponding T collision file, for all the collision file pairs, as needed. This includes removing the MR and MG files and removing the "_MT" file suffix. And 2) the user has done no other repo commands since the merge-out command on this project tree. (Commands on other project trees for the same team project can be done, and should not cause a problem.) For example, the 3 "fred_M*.java" files will be used to create a merged "fred.java" file (without the _MT) and the _MR, and _MG files will be removed – all by the user.

Merge-in is merely a duplicate of the check-in command, except that it has a different command name, and because no other commands have been run on the project tree, the merge-in command will always appear as the child of a merge-out command in the repo's set of manifest files. The merge-in manifest contents are equivalent to the check-in manifest file. (Note, if you don't want to create a merge-out manifest file, you must change your "grandma" detection mechanism appropriately.)

DAG

Because the mom-kid (parent-child) relationships among the manifest files can be a DAG (directed acyclic graph), finding the grandma snapshot (the most recent common ancestor) of the source and the target is a bit more complicated than finding the common ancestor in a tree.

If, in searching upward from the source or target snapshot you encounter a merge-out and merge-in pair of snapshots, you will have to follow both parent branches upward toward the root (the create-repo snapshot) because the grandma could be in either one of them. A simple way of doing this is to pick one branch to do immediately and put the other mom-branch's snapshot (or whatever is needed to identify and use it later) on a "pending" list (or array, stack, etc.). You will find a grandma candidate snapshot along every path from your (R or T) snapshot up to the root (create-repo snapshot). It is only the most recent of them that is the actual grandma snapshot for this R-and-T merge.

Dot-Files

If you find it convenient, you can put VCS files in the user's project tree at its root folder. These files must have a filename beginning with a dot/period, '.', and the prefix "vcsx" for our VCS system. If you decide to do this, make sure that dot-files are not included when you check in a snapshot.

User Interface

Make sure that you can enter a "command line" in a VCS web page edit box to initiate a command. The command line should include the VCS command name and all arguments needed to process that command. This is in addition to any fancier GUI mechanisms you have provided. All VCS command should be able to be run via this command line mechanism.

CECS-543 — Adv SWEngr — VCS Project 3

Team

The team size and naming style is the same as before, but you can change team members from the previous project if you wish.

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). Also, provide at least one test case where one of the T or R snapshots has multiple root paths.

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 – ugly code that is both hard to understand and complicates future modifications. 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; which is to say that your team's 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.

Project Reports

Same as before.

Submission & Readme

Same as before.

Grading

Same as before.