CECS-543 Adv SWEngr — VCS Project 2

VCS Project 2— Check-Out, Check-In, List, & Label

Introduction

This project is to build the second part of our VCS. In this project part, we add four new features: check-out, check-in (mostly already done), listing the existing manifest files (or a portion that fits in the display view), and labeling, again in **HTML+Javascript+Node+Express**.

For background material on actual modern VCSs, review on-line user documentation for Fossil, Git, and/or Subversion, etc. Note, in the terminology of a VCS, an "**artifact**" is a particular version of a file; multiple versions of the same file are artifacts of that file. Recall, a manifest file records the contents of a "snapshot" of a project-tree at creation, check-in, or check-out.

Label Command

The labeling feature allows the user to associate a label (a text string) with a given manifest file, in order to make it easier for the user to remember and identify a particular project-tree snapshot when issuing commands to our VCS. The user should be able to associate at least four (4) different labels to any given manifest file. (More is okay.) We can presume that the user is nice and always supplies a unique label – so we don't have to check for the label already existing in some other manifest file. A label is supposed to uniquely identify a manifest file. We can also assume that a label (a string) is at most only 30 characters long. (But you can handle longer labels if you wish.) To add a label to a manifest file, the user uses a Label command, and along with a label string argument he/she must also specify the VCS repository location and either the target manifest's filename or an existing label that is already associated with that manifest. Once a manifest is labeled, the user can refer to the manifest by that label name in any other VCS commands in place of using a manifest name.

Check-Out Command

The check-out ability lets a user recreate a specific version (snapshot) of the project tree. They do this by selecting a particular manifest file in the repo, as an argument. Of course, a manifest file specifies every version of every file from a particular version of a project tree. Note that a given repo folder only deals with one project (e.g., snapshots only for the Skyrocket project, or the Red-Bunny project, or the Halo project), but the repo can contain many versions (snapshots) of that one project. A snapshot can be created by anyone who has previously checked out a version of that project (but you do not have to verify this). On check-out, the recreated project tree is installed in an empty folder, which the user also selects as a second argument. We can assume that the target folder is empty. The check-out command also creates a new manifest file, of the checked out version, in the repo. The user should be able to specify the manifest file using a label, if it has one.

Check-In Command

The check-in ability lets the user update the repository (repo) with new version (snapshot) of a project tree for this repo. This means the VCS must add into the repo all changed files from the source project tree. So, each check-in is a (potentially) different "version" of the project tree, and you create for it a new manifest file. This allows the user to track the modification history from a given project tree back, through various project versions, all the way to the repo's creation; by examining the repo's manifest files. Note that we assume labels are forever (the user doesn't remove a label). The user's folder containing a version of the project tree that the user specifies as an argument to the check-in command should have earlier been the target of a check-out command (or was the original create-repo

project tree folder), and we will assume that this is always true. Therefore, in the repo's manifest files, we can trace from a given check-in (from a user's folder) back to the original check-out into that user's folder from a snapshot of some other user's project tree in some other folder, etc., all the way back to the original create-repo command. Note that your manifest files should reflect this ability, as it will be needed later. Also, note, a given repo only contains project-tree snapshots of a single project, which might be being worked on by several project members, each with one or more project-trees of their own.

Check-In Notes

Note that almost all of the check-in code has already been developed for the previous Create Repo project part. A few new issues must be handled:

1. For a newer artifact (version of a file with the same project folder relative path) in the repo, the leaf folder will already exist with an earlier version of that file having a different artifact ID. You merely have to add the newer artifact (file version) into the same leaf folder with its own artifact ID name.
2. If a project file has the same computed artifact ID (and project folder relative path) as an artifact in its leaf folder in the repo, then we can presume that the project-tree artifact and the leaf-folder artifact is the same file in both placed. So, you do not need to copy this project-tree artifact and overwrite the existing identical copy in the repo; but if that seems easier then you can do such an overwrite spending the extra copy time.
3. Also, you will create a "check-in" manifest file for this command. It will include the command and its arguments as well as the usual manifest information (same as for a "create-repo" command.) Note, if your project #1 manifest didn't include the "create-repo" command and arguments that was used to create it, please upgrade so that that manifest includes these.
4. Regardless of whether a project-tree file has been changed (ie, it's a new artifact ID) or not (ie, it's a duplicate artifact ID of an artifact in the corresponding repo leaf-folder), the file-name and its artifact ID must be recorded in the new manifest file for this check-in (with the check-in command line arguments and the date-time stamp, of course).

Check-Out Notes

For check-out, the user supplies the repo folder name and an empty target folder name and also selects a manifest (representing the specific version/snapshot of the project-tree files desired). The selection can be either by label or by a manifest filename. New issues must be handled:

1. You will create a new project tree inside the empty target folder. The files/artifacts copied from the repo must be those mentioned in the selected manifest.
2. Each needed repo file has an artifact ID as its filename and sits in a sub-folder (the leaf-folder) that is named with its project's filename. This repo artifact file will get copied into the empty target folder's new project tree in the correct relative folder path position and with its correct project-tree filename. For example, we should be able to recreate a project tree if we executed the command sequence:
   a. Create-repo
   b. “Accidentally” destroy/remove our project tree (outside the repo)
   c. Check-out to the old now-empty project-tree container folder by selecting the repo's create-repo manifest file
3. Also, you will create a "check-out" manifest file for this command. It will include the check-out

command and its arguments as well as the date and time and a line for each file checked out (just like the create-repo manifest).

Listing Command

This may not need to be a command. You should display the existing manifest file names and their labels. If the list is too large for the display, you should display a portion of the list. (Whether paging or scrolling or some other method is up to you.)

Testing

Test that the code to implement the Check-in and Check-out works. To do so, do a create-repo of your project #1 code project-tree. Then make mods and check-out to a new path location. Make mods and check-the modified project-tree in to the repo. Verify that the correct repo and project-tree changes have been made, etd.

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.

Also, check that you can add the labels “Alice 1” and “Bob #2” to a manifest and then that you can select that manifest for check-out with either label as well as by specifying its filename.

Team

The team size is the same as before, but you may change team members from the previous project if you wish.

Project Reports

As before. And please use this naming style: the **date in YYMMDD** format, , your **3-4 letter team name** (eg, `ABX`): like this “`543-p2-ABX-Standup-190212.pdf`”, **lest points be taken off.**

Readme File

As before.

Academic Rules

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

Submission

As before.

Grading

As before.