CECS-343 SWEngr Intro — 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 because it is like a clone of Create-Repo), listing the existing manifest files (or a portion that fits in the display view), and labeling. Again, this is to be done in **HTML+Javascript+Node+Express**.

Note, in the terminology of a VCS, an "**artifact**" is a particular version of a file. We will **really** need artifacts as our repo accumulates multiple snapshots of a project. We will need to store multiple versions (artifacts) of the same file. Because we will need to recreate (via the new VCS Check-out) command) a repo-owned snapshot in a new empty project-tree folder, we need to know all the files involved, and for each such file we need to know which particular version belongs to the snapshot that the user is requesting. The way we know about this is that every snapshot has an associated manifest file; this is the point of a manifest file.

Note that a given repo folder only deals with one project (e.g., snapshots only for the Skyrocket project, or snapshots only for the Red-Bunny project, or snapshots only for the Halo project), but the repo can contain many versions (snapshots) of that one project. This is to allow one person to save different useful copies of a project under construction, or to allow multiple people (team members) to simultaneously work on different parts of the project but to save their snapshots in one communal locations (the project's repository). (To control two different projects, use two repos, etc.)

For background material on actual modern VCSs, review on-line user documentation for Fossil, Git, and/or Subversion, etc., as needed.

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 [no frills MFS]. We can also assume that every 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 "new" 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 (i.e., the command is completed), the user can refer to the manifest by that label name in any other VCS commands in place of using a manifest name.

The VCS webpage should be able to show/list the available manifest files and their labels, so that the user can see what is available (and maybe even cut/paste from the list). The Label command should allow the user to create/specify each of the three arguments involved; whether by constructing a command line or some other method is up to you. (The Label command does not generate a manifest file.)

Check-Out Command

The check-out ability lets a user recreate a specific version (snapshot) of the project tree from the repo in a new empty folder. They do this by selecting a particular manifest file in the repo, as an argument (e.g., by typing its name, or one of its labels if any). Of course, a manifest file specifies

every version of every file from a particular version (the snapshot) of a project tree, along with the check-out "command line" and a date-time stamp. 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 [no frills MFS]. The check-out command also creates a new manifest file, of the checked out version, in the repo (which is intended to record this new project tree "development" folder). 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 repo with new snapshot of a project tree (for the project the repo was built – but we don't need to verify this [no frills MFS]). This means the VCS must add into the repo all changed files from the source project tree. (You can, of course, overwrite files in the repo that haven't changed from those in the project tree – it's a bit slower but your choice [no frills MFS].) So, each check-in is a (potentially) different "version" of the project tree, and you create for it a new manifest file that represents every file existing in that project tree (copied into the repo or not), along with the check-in "command line" and a date-time stamp.

This allows the user to track the modification history from a given project tree back, through various project versions (and maybe through various project tree folders), 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).

Note, 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 [no frills MFS]. 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.

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. If a project file has the same computed "CPL" artifact ID as an artifact in the repo, then we can presume that the project-tree artifact is the same file in both places. So, you do not need to copy this project-tree artifact and overwrite the existing identical artifact copy in the repo – but if that seems easier then you can do such an overwrite spending the extra copy time.
2. 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.
3. 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 repo), 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 selectied manifest 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. 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 from an original project-tree development folder
   b. “Accidentally” destroy/remove our project tree (outside the repo) development folder
   c. Check-out to the old now-empty original project-tree development 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.
Readme File As before.
Academic Rules As before.
Submission As before.
Grading As before.