CECS-343 — SWEngr Intro — VCS Project 2

VCS Project : Part 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** a snapshot to recreate a new copy of that snapshot's project tree, **check-in** a snapshot from an existing project tree (mostly already done because it is like a clone of **Create-Repo**), **listing** the existing manifest filenames (and for each, their user-defined labels) in the UI display view, and **labeling**, that adds a user-defined label to an existing manifest filename.

Again, this is to be done in **HTML+Javascript+Node+Express**, extending Part 1.

Note, in the terminology of a VCS, an "**artifact**" is a **particular version** of a file. We will **really** need artifact IDs (a unique ID for each file version) as our repo accumulates multiple snapshots of a project. We will need to store multiple versions (artifacts) of the same file, when each version is different from another (based on artifact IDs). Because we will need to recreate a project tree from a "repo-owned" snapshot in a new empty project-tree folder (via the new VCS **Check-out** command), we need to know all the files involved in the snapshot, and for each such file we need to know which particular version belongs to the snapshot that the user is requesting and what its folder path should be (created if missing) in the new project-tree. The way we know about this (i.e., the needed file versions and their folder paths) 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 (snapshots) 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 location (the project's repository) for others to check out. Also, this allows a team member to create an experimental version of the project & save snapshots of it in the same repo. (To control two different projects, use two repos, etc.)

List 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.) You may assume the list is never more than 25 manifest files in all (no frills). The **List** command **does not generate** a manifest file.

Label Command

The labeling feature allows the user to associate a user-defined 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. A user should be able to associate at least four (4) different labels to any given manifest file. (More is okay, but no frills.) 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). We can also assume that every label (a string) is at most only 20 characters long (no frills). (NB, we don't care if multiple users add labels to the same manifest, but 4 labels total is what we need to handle over all.) To add a label to a manifest file, the user uses a **Label** command, and along with a label string argument they 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. (They can see what the existing labels and manifest file names are via the List command.) Once a manifest is labeled (i.e., the command is completed), the user should be able to refer to the manifest by that label name in any other VCS commands that require a manifest as an argument.

The Label command arguments are the manifest (e.g., via an old label for it) to be labeled and the label string. (The Label command **does not generate** a manifest file.)

Check-Out Command

The check-out command is sometimes called a "clone" 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 – which will become the root of that recreated project tree. Users 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 (the snapshot) of a project tree, along with the check-out "command line" and a date-time stamp. On check-out, the recreated project tree is installed in the (assumed to be empty, no frills) target folder, which the user also selects as a second argument. 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 – possibly needed for later "merge" commands, not in Part #2). This new manifest file should be identical to the manifest argument file specified in the check-out itself, except for the command line and date-time stamp.

Check-In Command

The check-in ability lets the user update the repo with a new snapshot of a project tree (for the project the repo was built – but we don't need to verify this – no frills). This means the VCS must add into the repo all changed files from the source project tree. (You can, of course, add all files, overwriting files in the repo that haven't changed from those in the project tree – it's a bit slower but your choice [no frills.) So, each check-in is a (potentially) different "version" of the project tree, and for it you create 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.

A (check-in) snapshot can be created by anyone who has previously checked out a prior snapshot to that outside-the-repo project-tree (but you **do not** have to verify this – no frills).

The check-out & check-in manifest files allow 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; merely by examining the repo's manifest files. Note that we assume labels are forever (the user doesn't remove a label – no frills). So, if the user's label argument from the "command line" is saved in the manifest file (instead of the label's manifest filename) we can still trace back to the original create-repo project snapshot.

Note that for check-in, 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). 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 (originally) empty target folder from a snapshot of some other user's project tree residing 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.

CECS-343 — SWEngr Intro — VCS Project 2

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 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 stuff as for a "create-repo" command.) Note, if your project part #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 already 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 selected 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, which becomes the new project tree. 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. (Note, you may need to create sub-folders for the new project tree.) 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 a new empty** 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).

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 base project-tree. Then make mods and check-out to a new path location. Make mods and check-i n the modified project-tree in to the repo. Verify that the correct repo and project-tree changes have been made, etc.

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. (Alternatively, you can take screen pictures for this.)

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 the manifest filename.

Team

The maximum team size is the same as before, but you may change team members from the previous project if you wish (and you can change the team name).

Project Reports As before.
Readme File As before.
Academic Rules As before.
Submission As before.
Grading As before.