CS 543 — Adv SWE — SCM Project 2

SCM Project 2— Checked

Introduction

This is the second part of our SCM (Source Code Management) project. In this project part, we add three new features: check-out, check-in (mostly already done), and labeling.

The labeling feature allows the user to add a label (a text string) to a manifest file, in order to make it easier to remember. A manifest file must support up to 4 different labels at the same time. 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 ID a manifest.

The check-out ability lets a user recreate a specific version of the project tree. They do this by selecting a particular manifest file from the repo. The manifest specifies every version of every file required. The recreated project tree is installed in an empty directory, which the user also selects. The repo gets a new manifest file of the checked out version (for its records). The user should be able to specify the manifest file using a label.

The check-in ability lets the user update the repository (repo) with changed files in the project tree. Each check-in is a (potentially) different "version" of the project tree, and gets an associated manifest file created for it. This allows the user to track modification history from a given project tree back, through various project versions, all the way to the repo's creation. Labels are forever, so we assume the user doesn't change his/her mind later.

Team

The team may contain up to 3 members, and may be different from the project #1 team. Pick a three-letter name for the team based on the first letters of the members' last names. If fewer than three members are on the team, pick a third (and even a second) letter of 'X'.

Optionally, for teams, establish tasks and each day briefly discuss the Agile 3Qs: 1) What was completed yesterday?, 2) What is planned to complete today?, and 3) What obstacles are in the way?. Bread up a task so that some part can be completed each day. If you expect to have down (i.e., no-work) days, tell your team up front.

User Arguments

For both check-in and check-out, the user will supply a source folder and a target folder.

On check-out the source folder is the repo folder, and the target folder is an empty folder to receive a fresh copy of the specified version of the project tree. The project's root folder sits inside the target folder. In addition, on check-out, the user will select the manifest of the desired project tree (that was earlier checked in to the repo, or was the creation manifest).

On check-in, the source folder is an existing project tree top folder, and the target folder is the desired repo folder. (The user might be running several different projects, each with its own repo.)

On labeling, the user will select the manifest to label. We can assume it is unique and that the manifest has “room” for the label.

Check-in

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 artifact ID (and project folder path) as a file in the repo, then we can presume that it is the same file in both placed. So, you need not use this project file to overwrite the existing copy in the repo, but you can overwrite if that seems easier.

2. If a project file has the same file-name (and project folder path) as a file in the repo but has a different (new) artifact ID, then it is a different (new) version (it has different file contents) of the project file into the repo. You must copy this different project file into the filename sub-folder, so that both the old and new versions are present in that sub-folder. This will result is that repo sub-folder containing more than one version of the project file. (This is the purpose of the sub-folder being named with the project file's file-name.)
3. Regardless of whether a project file has been changed (ie, new artifact ID) or not (ie, old duplicate artifact ID), the file-name and its artifact ID must be recorded in a new manifest file for this check-in (with the check-in command line arguments and the date-time stamp, of course).
4. 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.

Check-out

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

1. You will create a new project tree inside the empty target folder. The files 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 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 position and with its project 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
   c. Check-out to old now-empty project-tree folder by selecting the repo's creation manifest
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, as follows:

1. On the prior minimal ptree and file:

```
mypt/
  hx.txt  // Contains the letter "H"; hence, you know the checksum.
```

2. On the prior tiny ptree with files:

```
mypt2/
  hx.txt  // As above.
  Stuff/  // A sub-folder
    hello.txt // Contains one line: "Hello world".
    goodbye.txt // Contains two lines: "Good" and then "bye".
```

3. On a small ptree with a sub-folder (called "src") and files:

```
mypt3/
  hx.txt  // As above.
  src/  // A sub-folder.
    main.fool // Main pgm file, Foo Language.
      // Contains one lines:  "Defoo main, darn sock."
    darn.fool // Weird hole-filler component file.
      // Contains two lines:
      // 1. "Defoo darn stuff: set thread.color to param.color;"
      // 2. " find hole; knit hole closed."
```

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.

Readme File

You should provide a README.txt text file that includes the class and section, your (team) name, the project/program name, instructions for building, instructions for use, any extra features, and any known bugs to avoid. Be clear in your instruction on how to build and use the project by providing instructions a novice programmer would understand. Please note that the instructor may not have your IDE available. If there are any external dependencies for building, the README must also list them and how to find and incorporate them. Usage should include an example invocation.

A README would cover the following:

- Program name
  - Your Name (authors, team)
  - Contact info (email)
  - Class number, Section (eg, 01),
  - "Project Part" and its number
- Intro (see the Introduction section, above)
- External Requirements
- Build, Installation, and Setup
- Usage
- Extra Features
- Bugs

Academic Rules

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

Submission

Same as for project #1.

Grading

Same as for project #1.