CS 543 SWE — SCM Project 1

SCM Project 1 — Create Repository

Introduction

This project part is to form a development team and to build the first part of our SCM (Source Code Management) project (AKA a VCS: Version Control System), in Java. This part only implements an initial use-case: Create Repo. It also makes a number of simplifying assumptions in order to get to working S/W quickly.

For background material, review on-line user documentation for Fossil, Git, and/or Subversion. Note, and "artifact" is a version of a file; if you modify a file, we call it a new artifact.

The SCM repository will potentially hold multiple versions/artifacts of multiple files; hence, the original file name is not sufficient to distinguish between two of its artifacts; hence, within the SCM repository (the location of all checked-in versions of files for a project) we will use a code name for each artifact and will put all the artifacts of a particular file in a folder which has the original file's name.

Team

The team may contain up to three members. 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 letter of 'X' for a missing team member.

Use Case

Title: Create Repository
Tag-line (AKA One-liner): Create a repository for the given project source tree.
Summary (AKA Contextual): We create a repository (repo) in the given target file path from a project source tree in the given source path. We are given the source and target. The entire source tree (including its root folder) is replicated in/under the repository folder. Additionally, a manifest for the command is created listing the command particulars, the date and time of the command, and a line describing each source tree file (AKA artifact) in the project at the time. Because we expect, **eventually**, to store more than one version (artifact) of each source tree file, we put the artifact under a leaf folder, where the leaf folder gets the file's name and the artifact gets an artifact ID (a code name). The leaf folder appears in the repository in same relative position as its corresponding file appears in the project source folder.
Simplifying assumptions:

1. All files in the project tree (ptree) will be included. (No exception black-list.)
2. No frills: We ignore user input mistakes.
3. A file version (AKA artifact) will consist of the full file. (No need to create deltas/diffs.)
4. The repo will include the ptree folder hierarchy.
5. Each ptree file will get a "leaf" folder of the same name to hold that file's artifacts. Thus, if ptree folder xcp/ has two files fred.c and jack.c, the repo will have folder xcp/ as well as leaf sub-folders fred.c/ and jack.c/ – where leaf folder fred.c/ will contain all that ptree file's fred.c artifacts and leaf folder jack.c/ will contain all that ptree file's jack.c artifacts.
6. We will create an artifact ID (AID) code name as discussed below.
7. The artifact (file version) that is in a leaf folder gets named by it's AID code name.
8. Assume that both source tree and target repo folders exist and that free disk space is adequate.
9. A command-line interface is sufficient.
10. Assume the source, and the target repo folders have already been created and repo is empty.

Artifact ID (AID) code names

The code name will be a rolling 5-byte weighted checksum of all the characters (bytes) in the file followed by a dot and the integer file size, followed by the file's extension. The 5 weights by which each 5 character group are multiplied are 1, 3, 7, 11, and, 17. Thus, if the file contents is "HELLO WORLD", the checksum S is:

S = 1*H + 3*E +7*L +11*L + 17*O + 1*' ' + 3*W + 7*O + 11*R + 17*L + 1*D;

and the file size is 11. Note, the ASCII numeric value of each character is used and we indicated the space character by ' '. For example if this was the contents of a version of file fred.txt, then the AID code name would be "`6098.11.txt`", in a leaf folder named "`fred.txt`".

Create repo

What needs to be done (a rough Main Scenario; AKA "Success Path")

The project tree will be copied into the repository, except that the project tree leaves, the files, will be handled specially:

Get the source tree path and the target repo folder path from the user.

Create a copy of the project tree folder hierarchy in the target repo folder.

Each project tree file will be copied into its own new leaf folder.

The new leaf folder will be given the file's name, including its extension.

Copy the file into its new leaf folder. The copied file's new file name will be a code name (artifact ID) but with the file's original extension unchanged.

Add an activity folder to the repository.

Create a manifest file for the repository creation (a record of this command's activity) containing

1. The "XIV" name, the "code name" of this SCM project.
2. The repository creation date time
3. The user command line given by the user
4. The full source path to the original project tree (e.g., C:/my/ptree)
5. The full target path to the repository folder
6. A line describing each project tree file copied, including: the artifact ID, the file's original name, and path in its project tree

Testing

Test that the code to implement the Create Repo use-case works

1. On a minimal ptree containing one file:
```
mypt/
  hx.txt  // Contains the letter "H"; hence, you know the checksum.
```
2. On a tiny ptree containing an extra folder with three files 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".
```

CS 543 SWE — SCM Project 1

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. 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

Submission

Your submission must, at a minimum, include a plain ASCII text file called **`README.txt`** (e.g., title, contact info (of all team members), files list, installation/run info, bugs remaining, features added) all necessary source files to allow the submission to be built and run independently by the instructor. [For this project, no unusual files are expected.] Note, the instructor doesn't necessarily use your IDE or O.S.

All source code files must include a comment header identifying the author, author's contact info (please, no phone numbers), and a brief description of the file.

Do not include any IDE-specific files, object files, binary **executables**, or other superfluous files.

Place your submission files in a **folder named** `X-pY_lastname-firstinitial`. (If working in a team, use the team's 3-letter acronym: `X-pY_RAK`, for team RAK.) Where X is the class course number (e.g., 123 for course CS-123) and Y is the project number (eg, 9 for Project #9) For example in CS-123 for Project #9, if your name were Alice Cooper, then you should use this:

```
123-p9_Cooper-A
```

Then zip up this folder. Name the .zip file the **same as the folder name**.

Turn in by 11pm on the due date (in the bulletinboard post) by **sending me email** (see the Syllabus for the correct email address) with the zip file attached. The email subject title should also include **the folder name**. [NB, If your emailer will not email a .zip file, then change the file extension from .zip to .zap, attach that, and tell me so in the email.] Please include your name and campus ID (for each team member) at the end of the email (because some email addresses don't make this clear). If there is a problem with your project, don't put it in the email body – put it in the README.txt file. Do not provide a link to Dropbox, Gdrive, or other cloud storage.

Grading

- 65% for compiling and executing with no errors or warnings
- 20% for clean and well-documented code
- 10% for a clean and reasonable **`README`** file
- 5% for successfully following Submission rules