CECS-543 Adv SWEngr — VCS Project 1

VCS Project 1 — Create Repository
Introduction

This project is 1) to form a development team and 2) to build the first part of our **VCS** (Version Control System) project. This first part only implements an initial use-case: Create Repo (Repository). It also makes a number of simplifying assumptions in order to get to working S/W quickly.

This project will be build 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 file are artifacts.

The VCS repository holds copies of all artifacts (i.e., all versions) of each file of a project "under configuration control". A file name alone is not sufficient to distinguish between several of its artifacts/versions; hence, within the VCS repository we will use a code name for each artifact and will put all the artifacts of a particular file in a **folder**, and that folder is named using the original file's name.

Note, this project will form the basis for the next project, so apply Rule #5 (Clean) as appropriate.

Team

The team is from two to four members. Pick a 3-letter name for your team (e.g., "ABX").

Use Case
Title: Create Repository
Tag-line: Create a repository for the **given** project source tree (including all its files and their folder paths) within the project.
Summary: The user needs to keep track of various snapshots of their project. Each snapshot includes the current state of each file in their project tree at a specific moment during project development. In order to keep track of each snapshot, we create a **repository (repo)** in **the given target folder** and copy a snapshop of the source tree in **the given source folder**. The entire **source** project tree **folder** (including its root folder) is replicated within (and immediately under) the **target** repository root **folder**. Additionally, on creation, a **manifest** (i.e., a snapshot summary) for this command is created listing the command particulars (i.e., the command line used), the date and time of the command, and for each project source file a line describing that source file (AKA that artifact) in the project along with its project folder's relative path. Because we expect, **eventually**, to store more than one artifact of each project file, we put the first artifact of a file under a new (non-project) leaf folder, where the leaf folder is given the file's name and the artifact gets an artifact ID (a code name). Note the contents of the artifact file is the same as its corresponding project source file snapshot. The leaf folder appears in the repository in same relative position as its corresponding file appears in the project source folder. The artifact ID format is described below.
Simplifying assumptions:

1. All files in the **project tree (ptree)** will be included. (No exception, black-list.)
2. No frills: You may ignore user input mistakes.
3. A file artifact will consist of the full file contents. (No 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 (initially just the first artifact). 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 (ArtID)** code name as discussed below.
7. The artifact (file version) that is in a leaf folder gets named by it's ArtID code name.
8. Assume that both given source and empty target folders exist, and that disk space is adequate.
9. A command-line interface within a web page (e.g., edit boxes & "Create" button) is sufficient.

Artifact ID (ArtID) code names

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

S = 4540 = 1*H + 7*E +3*L +7*L + 11*O + 1*' ' + 7*W + 3*O + 7*R + 11*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 this version of the source file **fred.txt**, the AID code name would be "`4540-L11.txt`", in a leaf folder named "`fred.txt`".

Modulus: Because the sum can get rather large for a big file, make sure the sum never gets too large by wrapping it using the following prime modulus operator: m == (2^31) - 1 == 2,147,483,647.

Project Reports

User Scenario: Write up a User Scenario which describes what the user wants the VCS to do, as an assistant to the user, and why the user wants that. Focus on the kinds of requests the user might make to the VCS. Also, indicate which features your team would like to implement (if time permits).

Feature/Tasks (WBS & Backlog): What is your current estimated hierarchy of tasks (WBS)? Indicate the task to sub-task relationships, if any. Note that these can be changed during project development. When modified, include it in a zip with the next Standup Status report. Give each task a number (and don't reuse numbers if you add/delete tasks).

Progress Board: Which WBS tasks have been A) Begun, by whom? B) Completed (and can be demonstrated) by whom, when? C) Verified (QA'd) by whom, when? Include the current Progress Board file at the end of each Standup Status report.

Standup Status, twice weekly. The Standup Status Report is due Sunday by 11pm and on the day before the last class session of the week at 11pm. One report per team. It should contain, for each team member, their Standup answers to the 3 standard Standup questions: Q1. What task(s) have you completed since last status? Q2. What task(s) do you plan to complete by next status? Q3. What obstacles are blocking your progress (on which task(s))? Consider sub-dividing a WBS task into "Half-Day" sub-tasks so as to be able to have a task completion (or more) for each status. Note that if a team member takes on a WBS task and sub-tasks it, then the sub-tasks should be added to the WBS and shown on the Progress Board. Also, note that keeping track of tasks, new/old sub-tasks, and completions takes effort, but can be streamlined. Don't forget that merging your completed task code into the team's mainline code requires integration testing. Tasks that have gone through QA should be demonstratable in class.

These documents should be delivered as **.pdf** files, and each filename should include your course number, your team name, the doc type (Uscen, WBS, Progbrd, Standup), and the date in YYMMDD format. E.g., "`543-ABX-Standup-190212.pdf`".

CECS-543 Adv SWEngr — VCS Project 1

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 string "Hello"; 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".
```
3. On your main source code project tree of files.

Readme File

You should provide a README.txt text file. 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:

- Class number
- Project name
- Team name and members
- Intro (use the tag line, above)
- Contents: Files in the .zip submission
- External Requirements (Node, etc.)
- Setup and Installation (if special)
- Sample invocation & results to see
- Features (both included and missing)
- Bugs (if any)

Academic Rules

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

Submission

All Necessary Files: Your submission must, at a minimum, include a plain ASCII text file called **`README.txt`**, all project documentation files (except those already delivered), 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 not use use your IDE or O.S.

Headers: 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.

No Binaries: Do not include any IDE-specific files, object files, binary **executables**, or other superfluous files. We don't use your IDE, your O.S., or your make/model of CPU.

Project Folder: Place your submission files in a **folder named** `X-pY_teamname`. Where X is the class number and Y is the project number. For example, if your team name is ABC and this is for project #2 in class CS-123, then name the folder `"123-p2_ABC"`.

Project Zip File: Then zip up this folder. Name the .zip file the **same as the folder name + ".zip"**. Turn in by 11pm on the due date (as specified in the bulletin-board post) by **sending me email** (see the Syllabus for the correct email address) with the zip file attached. The email subject title should include **the folder name**.

ZAP file: If your emailer will not email a .zip file, then change the file extension from .zip to .zap, and attach that.

Email Body: Please include your team members' names and campus IDs at the end of the email.
Project Problems: If there is a problem with your project, don't put it in the email body – put it in the README.txt file.
The Cloud: Do not provide a link to Dropbox, Gdrive, or other cloud storage. Note, cloud files (e.g., G-drive) are not accepted.

Grading

- 75% for compiling and executing with no errors or warnings
- 10% for clean and well-documented code (Rule #5(Clean))
- 10% for a clean and reasonable documentation files
- 5% for successfully following Submission rules