CECS-343 S/W Engr'g Intro — VCS Project 1

VCS Project : Part 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 [– AKA **SCM**, S/W Configuration Mgmt (Ch 22) – AKA **CMS**, Configuration Mgmt 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 built in **HTML+Javascript+Node+Express**. This infrastructure will be discussed in lecture, and a code sample will be provided. [CF, MERN "stack" on the web]

For background material on actual modern VCSs, review on-line user documentation for Git, Fossil, and/or Subversion [AKA SVN]. Note, in the terminology of a VCS, an "**artifact**" is a particular version of a file; multiple **different versions** of the **same file** (same based on filename and path) are called artifacts.

The **VCS repository** holds copies of all artifacts (i.e., all versions) of each file that belongs to a single multi-author project. Creating this repository for a project is called "**putting the project under configuration control**".

A file name alone is not sufficient to distinguish between several of its artifacts/(different versions); hence, within the VCS repository we will use a "code name" [an artifact ID "name"] for each artifact.

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

Team

The team is from two to four members. Pick a team name of 3 to 12 letters (e.g., "Groggy-Frog", or "ABX") – you can also use digits and hyphens. For the next project, you can change teams and team names. **["Team" means members help each other, otherwise you're just a group.]**

User Scenarios

Initial User Scenarios (discussions with the user) will be provided. Use cases can be built from these as well as further discussion with the 'users.'

Use Case (UC) – Create Repository (OneVerb+Object(s) format, from **the UC Tag-line**)
Tag-line (from the UC Summary): Create a repository in an empty folder from the given project source tree (including a "snapshot" of "all" its files, and a manifest listing them).
Summary (from the User Scenario, embellished with extra detail**):** The users need to keep track of various snapshots of their communal project, 1) in case they have to **"rollback" to a previous good working copy**, or 2) in case they want to preserve several experimental feature versions (branches of mainline) of the same project, or 3) in case they want to work on a bug fix without touching the "mainline" project code (branches of mainline) until the fix is well-tested, or 4) in case they want to ship the product with a different mix of features to different customers (a diff mainline for each feature "cluster"), or 5) you want to ship your product for different operating systems (a diff mainline for each).

Each project source tree **snapshot includes** the current (version) contents of each file in their project tree at that specific **snapshot moment** during project development. In order to **keep track of each snapshot**, we want a repository (AKA "repo") in the given empty repo folder (where all the snapshots of the project will eventually reside) and we want a copy (the snapshot) of the initial developer's source folder – including all files in all sub-folders to all nesting levels.

The "all files" in the project tree should **exclude** "dot-files", their name starts with a period (e.g., ".xtool.rc").

Additionally, on creation of the repository (i.e., using the Create-Repo operation), a snapshot "**manifest"** (i.e., a summary of files in the snapshot) is created in the repo (as a "dot-file") listing 1) the command particulars (i.e., the "command line" equivalent used), 2) the date and time of the command, and 3) **for each project source file** a data item (e.g., a text line) describing A) that source file's artifact ID and B) its relative pathname in the source project tree. The artifact ID format is described below. The manifest filename should be in the "dot-file" format ".man-<int>.rc" (where the <int> integer is the number "1", and a copy of this manifest file should be placed both in the repo and in the source project tree root folder of the Create-Repo command.

Note that each completely different project (e.g., Video-Game project versus AI-Robot project) should be kept in its own repository – one repository per different development project.

But, what is the UC main role?; what is the UC main scenario (3-8 processing steps)?; and what are the agents & major data parts? (Not to mention, what CRC cards should we have; and their contents?)

Implementation Design: Artifact ID (Art-ID) code names – (but no UC main scen, or CRC cards)

Weighted checksum: The Art-ID (code name) will have 5 parts: "A-B-C-D.E", where "D.E is the original filename and extension, and "A" is two Hex characters representing the weighted checksum of bytes in the file folder's relative Pathname string in the source project tree, and "B" is four Hex characters representing the weighted checksum of the file's Length in bytes, and "C" is four Hex characters representing the weighted checksum of all the Characters (bytes) in the file. The hex strings should be truncated (keep only the low-order hex digits) on overflow. (All this is for VCS security.)

Example for the relative path-file "`bot/a/b/fred.txt`" with the one-line contents, without the quotes:

```
"Now is the time for all good squiddies."
```

where A is from "`bot/a/b/`", and B is the length of that one-line (39 bytes), and C is that line.

ArtID: "`5F-0027-612E-fred.txt`".

The weights by which each character in a string (or each character byte in a file) are multiplied are **3, 7, 11, and 7** in a "loop". For example, here is the start of the C calculation:

C = `612E` = 3*N + 7*o + 11*w + 7*space + 3*i + 7*s + ...

(Note, the ASCII numeric value character code of each character is used.)

C = `612E` = 3*78 + 7*111 + 11*119 + 7*32 + 3*105 + 7*115 + ...

Project Development Reporting

Standup Status Report, twice weekly. The Standup Status Report is due **Monday's** and **Friday's** by noon-ish. One report per team, **CC'ing the other team members**. It should follow the sample Standup Status Report format discussed in lecture. This document should be delivered **as a PDF file**. Note that the date section should be in YYMMDD: E.g., "`343-06-p1-ABX-Standup-210212.pdf`", for status on February 12th, a Friday.

Consider sub-dividing big tasks into "**Half-Day Rule**" sub-tasks so as to be able to have a task completion (or more) for each status – tasks for which completion is **easily demonstrable**.

Testing

Test samples will be provided.

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 and section
- 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-section and Y is the project number. E.g., "**343-03-P1-Bozobus**".

JS files: For each JS file, change its name by adding a ".txt" extension to it, producing "foo.js.txt". This is to workaround a few email systems that refuse to send a zip file containing JS files, if you are submitting by email.

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**.

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.

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