362 — SWE Fundamentals — VCS Project 1

VCS Project 1 — Create Repository

Introduction

This project is to form a development team and to build, in C++, the first part of our VCS (Version Control System) project (AKA an SCM: Source Code Management system). 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.

For background material, review on-line user documentation for Fossil, Git, and/or Subversion. Note, in the terminology of a VCS, an "**artifact**" is a particular version of a file; if you modify a file, we call the modified file a new artifact.

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

Note, this project will form the basis for the next project, so apply Rule #4 as appropriate.

Team

The team may contain from two to four members. Pick a three-letter name for the team. (If two teams pick the same TLA, I will disambiguate them.)

Use Case

Title: Create Repository

Tag-line (AKA One-liner): Create a repository for the given project source tree and all its files, including their folder paths within the project.

Summary: In order to keep track of all versions of all project files, 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 paths. The entire source tree (including its root folder) is replicated in (and immediately under) the repository root folder. Additionally, on creation, a manifest for this command is created listing the command particulars (i.e., the command line), 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 at the time along with its project folder relative path. Because we expect, **eventually**, to store more than one version (artifact) of each source tree file, we put the artifact under a (non-project) leaf folder, where the leaf folder is given the file's name and the artifact gets an artifact ID. Note the contents of the artifact file is the same as its correspond project source file. 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: We ignore user input mistakes; we provide no embedded help.
3. A file artifact (AKA version) will consist of the full file contents. (No deltas/diffs.)
4. The repo (repository) will include the ptree (project tree) 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.txt, the repo will have folder xcp/ as well as leaf sub-folders fred.c/ and jack.txt/ – where leaf folder fred.c/ will contain all that ptree file's fred.c artifacts (e.g., 4F89-L102.c) and leaf folder jack.txt/ will

contain all that ptree file's jack.txt 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 source tree and target repo folders exist and that free disk space is adequate.
9. A command-line interface is sufficient.
10. Assume the target repo folder and is empty.

Artifact ID (ArtID) code names

Weighted checksum: The code name will be a rolling 5-byte weighted checksum of all the characters (ASCII bytes) in the file (i.e., the bytes in the file's contents) followed by a hyphen and an "L" 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" (note the space character), the checksum S is computed as:

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 the sum, S, was 6448, with a file length of 11, for a version of file fred.txt, then the ArtID file name would be "`6648-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 number: m == (2^31) - 1 == 2,147,483,647.

Testing

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

3. 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 file.

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 number and name
- Team name and members
- Intro (including the algorithm used)
- Contents: Files in the .zip submission
- External Requirements (None?)
- Setup and Installation (if any)
- Sample invocation & results to see
- Features (both included and missing)
- Bugs (if any)

362 — SWE Fundamentals — VCS Project 1

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 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.

Project Folder: Place your submission files in a **folder named** `362-p1_teamname`. For example, if your team name is ABC then name the folder `"362-p1_ABC"`**.**

Project Zip File: Then zip up this folder. Name the .zip file the **same as the folder name**. 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, attach that, and tell me so in the email.

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 correctly with no errors or warnings
- 10% for clean and well-documented code (Rule #4)
- 10% for a clean and reasonable **`README`** file
- 5% for successfully following Submission rules