Assignment 1: A Simple Backup System

Due: Wednesday January 30, 11:59 p.m.

1 Introduction

Version control software is one of the most important tools used by professional programmers. It allows programmers to track changes made to source code and documentation, helps merge changes made by several programmers into the official source code tree, and holds the official current version of the software in the repository.

For this assignment you will write a simplified version control system, called sbs (Simple Backup System). Who knows, you may even decide to use sbs to backup your assignments, although you should learn to use a "real" version control system such as CVS.

2 Specifications

sbs is a single program that will execute one of 4 different commands given as command line arguments: init, add, commit, and update. The starter code includes the functions for init and add. Your assignment is to write commit, and update (and of course main).

While sbs is not a full-featured backup system, it should still be usable. This means that your program must include appropriate error and warning messages, and possibly an indication of what the program has done. Think about the messages you would like to see as a user. They should be short (no more than one line), but clear and helpful.

The titles of each of the following subsections use the same format as the Unix man pages. FILE refers to a regular file or a directory. Arguments appearing between square brackets are optional, and the ellipses (...) indicate that the argument may appear more than one time. Therefore, add takes one or more files or directories as arguments, while commit and update take zero or more files or directories as arguments. For example, the following are all valid invocations of the sbs program:

    sbs init
    sbs add file1
    sbs commit
    sbs commit dir1 file1

2.1 init

The init command ensures that the backup directory is created. The backup directory or repository is hard-coded in sbs as ~/SBS. All backed up files will be copied into ~/SBS or one of its subdirectories.
2.2 add FILE...

The add command adds one or more files or directories to the list of files to be backed up. Each directory that contains files to be backed up must also be added to the repository, and contains a hidden file called .sbs that has the list of files and directories that will be backed up. For example, in Figure 1 the files c and x have been added (and committed) to the repository.

2.3 commit [FILE]...

The commit command actually copies files to ~/SBS and its subdirectories. It copies only the files listed in the relevant .sbs file or files. If commit is given no arguments, it backs up files from the current working directory. Otherwise, it backs up each of the files or directories listed on the command line.

The directory structure inside the repository is determined by the first directory to be committed to the repository. For example, consider the path ~/a/b/c where ~/ does not contain a .sbs file, but a contains a .sbs that has b in it. When the commit command is run, it will expect to either find or create the directory ~/SBS/b. You do not need to handle the case when a is later added to the repository. See Figure 1 for an example.

When a directory argument is given to commit, it will backup all of the relevant files and directories in that directory recursively. If sbs commit ~/a/b is run using the example in the previous paragraph, then the relevant files in b and c will be backed up (assuming c is a directory).

To copy a file in C you need to read in the file and write the copied file. Because you don’t know in advance whether the file is in ASCII or binary, you should use fread and fwrite to copy the file.

Finally, each file (but not directory) in the repository will have a version number appended to its name. Each time a file is committed, a new copy of the file appears in the repository with the version number incremented. This enables multiple versions of the same file to exist in the repository. The function unique_name is given to help with this.
2.4 update [FILE]...

The update command brings the working directory up to date with the repository. It copies the most recent version of the relevant files to the appropriate locations in the current working directory.

If called with no arguments, update will update all files and directories in the current working directory. For each file, it will find the corresponding file in the repository with the largest version number and copy it (removing the version number) to the appropriate directory. The repository does not change.

Like commit, update is recursive if given a directory as an argument.

3 Further details

You will probably want to read the following man pages carefully (the numbers indicate in which section the man page is found): readdir(3), opendir(3), stat(2), string(3), index(3), mkdir(2), getcwd(3), chdir(2). Man pages are at first painful to read, but part of this assignment is to give you practice reading them. Often when you read a man page you will be looking for a particular piece of information: return value, a particular option, or the required header file. While preparing the solution for this assignment, I kept one window open for the main purpose of displaying man pages.

Do not rewrite library functions. This particularly applies to string manipulation functions.

Think about the structure of your program and the possible functions that may be needed before you write the whole program. It will save time and you will end up with a smaller, more readable program.

Do not make this assignment harder than it is. Your program should work correctly if the user uses it in a sensible way, and print out error messages for the obvious errors. Document your decisions (concisely) in comments in the code.

Hand in a program that compiles and does not crash under any circumstances.

What you are given

See the web page for the starter code and the Makefile. You should not need to change the code, but you may add functions to the files if appropriate.

The only changes you should make to the Makefile are to change the OBJS, SRCS, or HDRS definitions. You should study this file to make sure you understand how it works. A normal Makefile would not contain nearly so many comments. (You may change the print, tar, and clean rules, if you find it convenient.)

What to submit

Your assignment will consist a Makefile and several source files. Submit all files required to make your program, even the ones you were given.

All files you submit should have your name and student number at or near the top of the file. You will be graded on the quality of comments in your code, programming style, and correctness.

Check the web page for important instructions on submitting assignments.
4 Examples

Here's a running example of how one could use sbs. It does not demonstrate all of the features of the program. Suppose I have a directory called a1 in my home directory, and I want to back up some files in it. ($ is the shell prompt)

```
$ pwd
/u/reid
$ ls
a1/
$ sbs init
$ ls ~
SBS/    a1/
$ sbs add a1
$ cat .sbs
a1
$ ls ~/SBS
$ ls
Makefile db.c db.h db.o sbs.c sbs.o
$ sbs add db.h db.c sbs.c
$ ls -a
.sbs Makefile db.c db.h db.o sbs.c sbs.o
$ cat .sbs
Show the list of files that will be backed up
.sbs contains a1
$ ls ~/SBS
Note that there is nothing in the backup dir
Let's add some files
$ ls
Makefile db.c db.h db.o sbs.c sbs.o
$ sbs add db.h db.c sbs.c
$ ls -a
.sbs Makefile db.c db.h db.o sbs.c sbs.o
$ cat .sbs
Show the contents of .sbs
(The order doesn't matter.)
$ ls ~/SBS
Now there is something in the backup directory
a1
$ ls ~/SBS/a1
Look inside the a1 directory
$ ls ~/SBS/a1
Note that only the added files are committed
$ sbs commit
After editing the files commit again
$ ls ~/SBS/a1
Look inside the a1 directory
$ ls ~/SBS/a1
Now the latest version has been saved
```

sbs.c