Program #1
A Shell Script to Find Unnecessary Files
Due Wednesday, September 29, 2004

Name: _____

Lab Time: _____

Grade: _____/30

On my honor, as a University of Colorado at Boulder student, I
have neither given nor received unauthorized assistance on this work.
Signature: _____

Backup Files and Temporary Files

If you have been using emacs to edit text files, you have probably noticed that your directories look something like this:

```
program1 program2
program1~ program2~
```

Every time emacs makes a change to an already existing file, it creates a backup file. This file has the same name with a tilde `~` at the end. If you accidentally erase something that you wanted to save, then the backup file is there to recover it.

However, after a while, all of your directories fill up with these backup files. Eventually, you may want to perform some "house cleaning" and remove all of these backup files. It would be time consuming to go into each directory to find each one individually. As such, you can use the `find` program to search your directories for these files.

The same problem exists for temporary files. You may create files in your `tmp` directory, use them for a while, and then not need them anymore. For this program you will write a shell script that finds all of these unnecessary backup files and temporary files.

The Shell Script

Write a **tcsh** shell script that searches under a specified directory and prints out the **relative path** to each unnecessary file. The input directory will be given as a command line parameter when your program is run. Do not "cd" to the input directory as this will effect the definition of "relative path". Each unnecessary file should be printed on a separate line. Below we define exactly what qualifies as an unnecessary file:

- Any ordinary file under the input directory (including its subdirectories) whose name ends with `~`.
- For any directory named `tmp` under the input directory, locate all ordinary files within and under that `tmp` directory. Note: there can be more than one `tmp` directory located under the input directory. In addition, the input directory itself might be named `tmp`.
- **Do not** print out the name of any directory, even if it ends in `~`, or is contained in a directory named `tmp`. Only print out ordinary files.
- For simplicity, we will not test your script on nested `tmp` directories, or on files whose names end in `~` contained in a `tmp` directory. In short, you don't have to worry about a filename being printed twice.

For example, if the input directory is `.` (dot, the current directory) and contains `file1`, `file1~`, and `tmp`, and `tmp` contains `file2`, and `directory`, and `tmp/directory` contains `file3` then your shell script should print:

```
./file1~
./tmp/file2
./tmp/directory/file3
```

Saving Files. Occasionally there will be temporary files that are in active use and should not be listed by your shell script. To indicate these files, you can place the string `#save#` somewhere within the file.

Therefore, for a complete solution to this assignment, your shell script must print only those unnecessary files that match the criteria above and do not contain the string `#save#`.

To make your life easier, you may not want to implement this program all in one `find` command. You will be submitting a shell script, and you can run several commands in the script to create your final list of files to be printed out. Try creating commands that get you part of the way there, and store these partial answers in shell variables to be used later. In particular, a technique covered in Lecture 3 may be very helpful in that regard.

Special Shell Variables

Your shell script should be named `program01` It will be run with a single argument which is the directory in which to start searching. For example, we might run your shell script like this:

```
program01 ~/csci3308
```

The shell has special variables to handle command line arguments. The variable `$1` represents the first command line argument, `$2` the second, etc. They are used just like normal shell variables. You can try them out by writing a shell script containing:

```
#!/bin/tcsh -f
echo $1
```

.

There is another special variable which might be useful called `$status`. After the shell runs a command it sets the value of `$status` to indicate whether the command succeeded or failed. For example, if you run a grep command, and it finds one or more matching lines, `$status` is set to 0. If it does not find a match then `$status` is set to 1. You can check `$status` with an `if` statement to control your shell script.

Testing Your Program

In order to help you develop your program, we have provided two testcases that can be downloaded from the class website. The testcases are called `testcase1` and `testcase2` and contain a number of directories and files that may or may not meet the criteria given above for unnecessary files. Take a look at these directories and determine manually what files match the criteria (note, this is not a trivial task), then develop an algorithm that can produce the correct list of files.

In addition, you should make sure that your program does not crash if it is invoked in one of the following ways:

```
program01 program1test/testcase1

program01 program1test/testcase2

program01 program1test/testcase2/depth

program01 program1test/testcase2/empty

program01 program1test/testcase2/tmp

program01 program1test/testcase2/notmp
```

Note, we may also test your program by simply cd'ing to one of the above directories and invoking your program with the following command:

```
program01 .
```

Finally, we reserve the right to test your program on additional (private) testcases.

Turning in Your Program

Please print your shell script and bring it to lab on the 29th. In addition, email a copy of your program to your TA. (You can find your TA's email address on the class website; click on the "contact information" link. Your program should contain comments to explain what it is doing, and it should also contain the following header (updated with your contact information):

CSCI 3308 - Program 01 - Fall 2004
<Insert Your Name and Email Address Here>
<Insert Your Lab Section Here>

Any questions? Send mail to Dr. Anderson or your TA. **Under the CU Honor Code, you may not seek help from any other source, including other students and the World Wide Web, nor may you discuss the approach you are taking to solve this program with other students.** You may, however, use your "Linux Shells By Example" textbook.

Evaluation

A correct program handed in by September 29th will receive 30 points. Partial credit will be available. In particular, we will apply your script to a number of testcases. 5 points will be taken off of your score for each testcase that it fails. As such, it is possible to get 0 points on this assignment, if your script fails on six or more testcases.