Programming Assignment #2 Shell-level Process Management

Spring 2015

Deadline:

See **course calendar**.

Educational Objectives:

This assignment is intended to familiarize you with the shell view of process management. You will write a shell script that tests a program similar to the one you modified for your first programming assignment. The main effort will be learning the Unix "/bin/sh" (Bourne) shell language, and taking care to anticipate and catch all the ways that a process might fail. You will also be expected to verify that your shell script works the same on both Linux and Solaris/SunOS systems.

- Learn or review the process management features of the Unix (Bourne) shell, "sh".
- Become familiar with process management concepts, including: process creation, process termination, exit status, signals, resource limits
- Experience the importance of shell script portability.
- Become familiar with the concept of using scripts for automating the software build and (regression) testing process.

Deliverables:

Modified versions of the provided file *testone*, and a new shell-script file *testall*, plus any additional test programs that you have created.

Tasks:

1. Create a new working directory, *P2*, and copy into it gzipped tar-file `~baker/opsys/assign/P2/P2.tgz>P2.tgz`. Unzip and un-tar the file, using the command `tar xzvpf P2.tgz`. Test the installation, by executing `./testone sol0` and looking at the file that is created in the subdirectory *results*.
2. Before any coding, read the shell script *testone*, so that you understand it. Use the on-line man-page for *sh* and/or web-page tutorials on the Bourne shell if you run into features you do not understand.
3. Modify the script *testone* and create a new file *testall*, as follows:
    1. The present script only runs the tested program from one of the "sol*" subdirectories. Create a new script, named *testall*, that contains a loop that will execute *testone* for each of the subdirectories with names of the form *solNNN* where *NNN* is an integer. (Hint: Read about shell if-statements and for-loops.)
    2. If the tested program (*sim*) gets stuck in an infinite loop, as it will if you run `./testone sol2`, the log file can be very long. Modify *testone* so that it will truncate the output that goes to the log file, to a maximum of 1000 lines. (Hint: Read the man page on *ulimit*.)
    3. The present *testone* does not check that the output of the tested program is correct. Modify the script so that it compares the output of the tested program against the correct output from the baseline program in directory *sol0*. There is an example of a program with incorrect output in *sol3*.
    4. The present *testone* only runs the tested program with the default parameter values. Modify the script to run the tested program at least twice, using different command-line parameter(s).
4. Update the comments *testone* to include your name, the date of last modification, and an explanation of the changes you made (the new features you added to the script), and make certain you have appropriate header comments in file *testall*.
5. Check that the new script works, by running it on the four examples (*sol0*-sol3), and debug as necessary. Add at least one additional test case to cover other possible failure modes, using names *sol4*, *sol5*, *etc.*.
6. Now switch to the other operating system (to SunOS if you were working on Linux before, or *vice versa*) and

repeat the testing as above, correcting any problems that show up. If you make any changes, switch back to the other operating system and repeat, until you are sure your program works correctly on both systems.

Further Instructions:

- Do not modify any of the other files. Just modify *testone*, write *testall*, and add additional *sol** test directories.
- There are Unix utilities for counting the length of a file (*wc*), truncating a file, and comparing files (*cmp*, *diff*). You may call those in your script as appropriate. Try to find standard Unix utilities, which are available on both Linux and SunOS. If you cannot avoid differences, tailor your shell-script to handle the differences, using conditional constructs.

References:

- On-line notes from this course.
- SunOS and Linux man-pages.
- Other web pages, on the Bourne shell and Unix utilities.

Delivery Method:

First read the Study Guide section on **Submitting Assignments**, for general instructions.

1. Log into *shell.cs.fsu.edu*.
2. `cd` to the directory where you have your working solution, and make certain the version of the files *testone* and *testall* you want to submit and any *sol** test directories you have created are in that directory.
3. Execute the shell script `submitP2.sh` using the Bourne shell, by entering `sh submitP2`. If it works correctly, you should see a successful completion message, and eventually see the usual two confirmation e-mails.

Assessment:

Start by reading the **Grading Criteria for Programming Assignments** section of the Study Guide for general grading criteria.

Grading will be according to the following rubric. However, you can expect to have your score reduced if your code does not adhere to the instructions given in this file.

Criterion	**Points**
testall script correctly iterates over all the *sol** directories.	30
testone correctly handles problem #2 above (limits output).	15
testone correctly handles problem #3 above (checks correctness of output).	15
testone correctly handles #4 above (2 or more variations on command line parameters).	15
At least one additional test program provided, for other failure modes	15
Appropriate comments in test files	10

T. P. Baker.