CMPT 214: Programming Principles and Practice
Term 1 2016-17
Lab 2 - More LINUX/UNIX

Before starting this lab exercise, pay attention to the presentation by the lab instructor on starting X11 (XWindows) via *XQuartz*, starting an `xterm`, accessing tuxworld via `ssh` in that `xterm` window, and then running an XWindows client on tuxworld. Follow along on your workstation with the lab instructor, replicating what he does, and make sure that you end up with the same results.

Perform each of the tasks below in a virtual terminal window. For each task, copy-and-paste the contents of your virtual terminal window (including the commands that you typed, as well as any output produced by the commands you gave) into a text file called `lab2.txt`. However, do not include extraneous or superfluous commands or output; only include content relevant and essential to the specified task. Then, with a text editor, **add text and identifying information to clearly distinguish which commands/output/code correspond to each task.** Submit `lab2.txt` through moodle when done. The submission is due at 11:55 p.m. on Thursday, September 22. This lab is out of a total of 14 marks, with each question (1a, 1b, ..., 3e) being worth one mark except for 2(d) which is worth 2 marks. Unless otherwise specified, all commands should be run on tuxworld using the `bash` shell.

1. (a) Create a directory called `214lab2` under your home directory, and then `cd` to it. Use this new (sub)directory as your current working directory when completing all the remaining tasks in this lab.

   Use the `touch` command to create 15 (empty) files called `file1.txt`, `file2.txt`, ..., `file14.txt`, `file15.txt`. Also use `touch` to create five files called `data1.dat`, ..., `data5.dat`.

   The `touch` command is described on pages 985-986 of the Sobell text. Note that you can give multiple arguments to a single `touch` command.

   You may also be able save some typing by using "arrow keys" on your keyboard. With the "up arrow" key you can retrieve the last input line, and with the "right arrow" and "left arrow" keys you can position your cursor within the retrieved line so that you can make local edits to it. When you have completed your local edits, typing the "return" or "enter" key will cause `bash` to interpret your edited (input) line. Page 31 of the Sobell text gives more information.

   (b) Use a UNIX command involving filename wildcards to list all the files in your `214lab2` directory (from step 1a) that end in "`.dat`".

   (c) Use a UNIX command involving filename wildcards to show the permissions, owner, group, etc. for (only) the files `file10.txt`, ..., `file15.txt` (in your `214lab2` directory from step 1a).

   (d) Invoke the `file(1)` command on (only) the files `file1.txt`, `file3.txt`, `file4.txt`, `file6.txt`, and `file9.txt`. Use a single command involving filename wildcards to do this.

   If you need supplementary resource material on use of filename wildcards beyond what was given in class, you may find pages 148-152 in the Sobell text useful.

2. (a) Use a UNIX command to (try to) delete the file `fake_file.txt` in your current working directory (`214lab2` from step 1a). Do this step even though `fake_file.txt` does not exist. Your command must be structured such that, if the file exists, you are not prompted for confirmation, and if the file does not exist, no error message is printed. Consult the appropriate manual page to learn how to do this; i.e. consult a relevant `man` page or `info` page for the options you need.

   Submit your command and any output generated. Do not submit any logs from consulting `man` page or `info` pages.

   (b) Use a text editor or UNIX command to create a file called `my_name.txt` that contains your name and student number. (Log of using the editor does not have to appear in your `lab2.txt`.) Use a UNIX command to concatenate the contents of `my_name.txt` and `fake_file.txt` (which, as a result of the previous question, does not exist) and output the result to the default standard output.

   Note that an error message regarding the nonexistence of `fake_file.txt` should be produced.

   (c) Repeat the above command (step 2b), except redirect standard output to the file `standard_output.txt` and redirect standard error to the file `standard_error.txt`. Then show (i.e. output to the terminal) the contents of `standard_output.txt` and `standard_error.txt`.

   (d) Remove files `standard_output.txt` and `standard_error.txt`. Then execute the same command as in part 2c, except discard any output sent to standard error. In other words, the error output should not be printed to the screen or output to a file on disk, but rather redirected in such a way that it is discarded. Then show (i.e. output to the terminal) the contents of `standard_output.txt`. Finally, with an `ls` command, demonstrate that `standard_error.txt` does not exist (was not created by your command).

   If you need supplementary resource material on redirection beyond what was given in class, you can consult pages 133-141 and 282-284, among others, in the Sobell text.

3. (a) Run the program (command) `top`. Then suspend this process.

   (b) Run either the `env` or `printenv` command, and in the output identify (visually) the environment variable that contains the current user name. In `lab2.txt` you only need to have the output from the `env` or `printenv` command for this step.

   (c) Look at the manual page for `pstree` (with either the `man` or `info` commands) to find the option which will direct the command to display the PID of each running process. Give this option to `pstree`. In addition, provide as an argument to the command the expansion of the environment variable you identified in step (b). The result should be a hierarchical view of all your processes complete with PIDs. Locate the PID of the `top` process you started (and then suspended) in part (a). In `lab2.txt` show the `pstree` command you used and the output from that command.

   (d) Use a UNIX command to send signal 9 (`SIGKILL`) to your suspended `top` process. Obtain the necessary PID from the output in question 3c. After entering your command, type an extra "return" (or "enter" on some keyboards) character to the shell. You should get a message telling you that the `top` process has been killed.

   (e) Repeat the `pstree` command from step 3c and confirm that the `top` process is no longer running. In `lab2.txt` again show the `pstree` command you used and its output.

   The Sobell textbook has useful information on all the commands used in Question 3.