

Assignment 1

COMP 206, Fall 2015

Due: Sunday October 4th, (23:59) via My Courses

100 marks total

Please read the entire pdf before starting.

You must do this assignment individually. Read the question descriptions carefully and follow the filename conventions specified for your submission. We will mark your submissions using an automated script that depends on you using the right file names (including capitalization, use of underscores, etc). Pay close attention because the TA's are instructed to deduct 10% for submissions that do not follow the specification.

Getting Started:

The first step in preparing this assignment is to extract the problem files that you'll find in the zip attached alongside these instructions. The file is called “**206_A1_problems.zip**”. You should create a directory, save the zip file in that directory and extract it. For example

```
$ mkdir 206_assign1
```

```
$ cd 206_assign1
```

```
$ mv ~/Downloads/206_A1_problems.zip .
```

```
$ unzip 206_A1_problems.zip
```

```
$ ls
```

```
206_A1_problems.zip Problem1 Problem3
```

(*NOTE: There is intentionally no folder for Problem 2.)

You will have to provide both written answers and code. Written answers should be placed in a file named “**A1_written_solutions.txt**”. Code file names will be specified separately in each question.

Good luck, happy coding, and please come to TA or instructor office hours early on if you get stuck anywhere!

1. File Permissions

Part a) (5 marks) The provided “**Problem1**” directory contains a file named “**answer.txt**”, which is a text file that holds the answer. Try to show the contents of a file with cat. For example:

```
$ls Problem1/  
answer.txt  
$cat Problem1/answer.txt
```

(**Hint**, you are NOT supposed to see the contents of this file yet!)

Write a description of the outcome of this command in your solution text file. Your answer can be just a few words, but make sure to include the other commands that you ran in order to diagnose any failures along with their output.

Part b) (5 marks) Fix the problem with the “**answer.txt**” file so that you are now successfully able to use the “cat” command above to view the file contents. Copy the sequence of commands that you used and their outputs. Make sure to copy the contents of the file into your written answer, to show us it worked.

2. Process Monitoring and Control

Part a) (5 marks) The `ps` command can report many things about the jobs running on a system, but it can be confusing to understand its output. Run the command “`ps l | head`”, which will give you something like the this:

```
$sleep 60 &
[1] 11508
$ps l
F  UID    PID  PPID  PRI  NI     VSZ   RSS WCHAN  STAT TTY      TIME COMMAND
0 16906   8472   8471   20    0   38328  11488 wait    Ss   pts/8    0:00 -bash
0 16906  10460  10459   20    0   38328  11480 n_tty_  Ss+  pts/12   0:00 -bash
0 16906  11508   8472   20    0   12384    704 hrtime  S    pts/8    0:00 sleep 60
0 16906  11516   8472   20    0   15204   2100 -       R+   pts/8    0:00 ps l
$
```

Write, in your solutions file, a list ALL of the column headings. Give a brief description of the meaning of the information provided by that column.

Part b) (25 marks) The “kill” command is a useful way to terminate running processes, especially those that you started been started from your current terminal (ctrl-c will not do the job) Using “kill” within a shell script is something that should only be done carefully and cautiously because doing things incorrectly can have serious consequences. So let's try it!

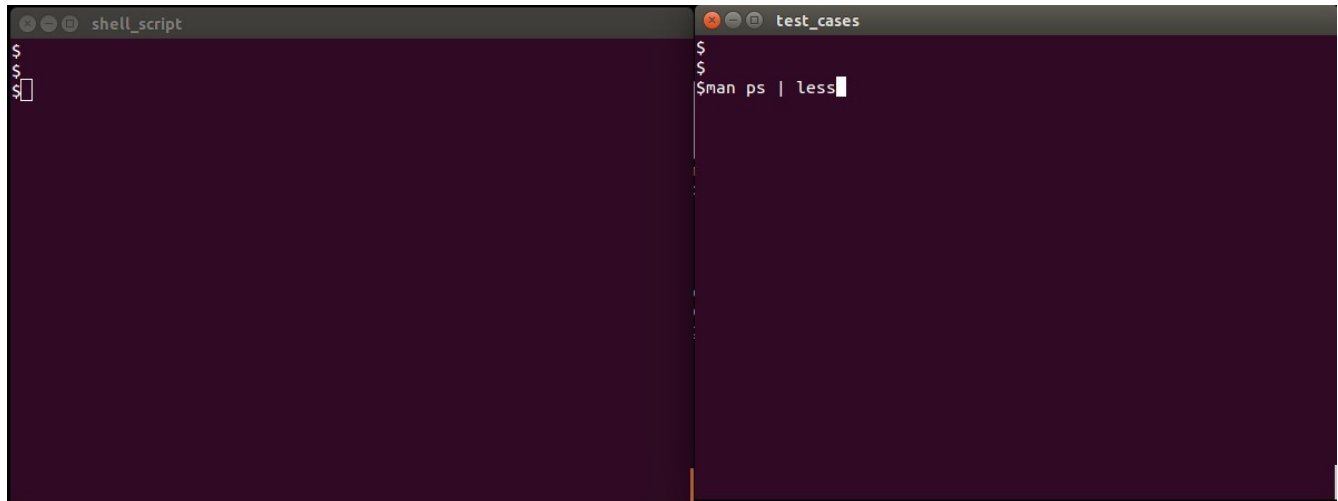
Write a shell script called “`q2b_less_less.sh`” with the following specifications:

- When executed, causes any currently running process of the “less” command started by your user account to be terminated.
- Returns rapidly (restoring the ability to run less).
- Does NOT terminate any other process, including processes that you own and especially not processes owned by other users.

Test this by starting two separate shell sessions connected to the same machine. This can be done by simply opening two terminal windows if you are in a Trottier lab or using a local Linux installation. If you are remotely connecting, make a second remote connection to the same host. Test by doing:

- In shell 1, launch a “less” command to view something (e.g. `$ man ps | grep less`).
- In shell 2, run your script (e.g., `$ sh q2b_less_less.sh`) -> the less viewer in shell 1 should die immediately.

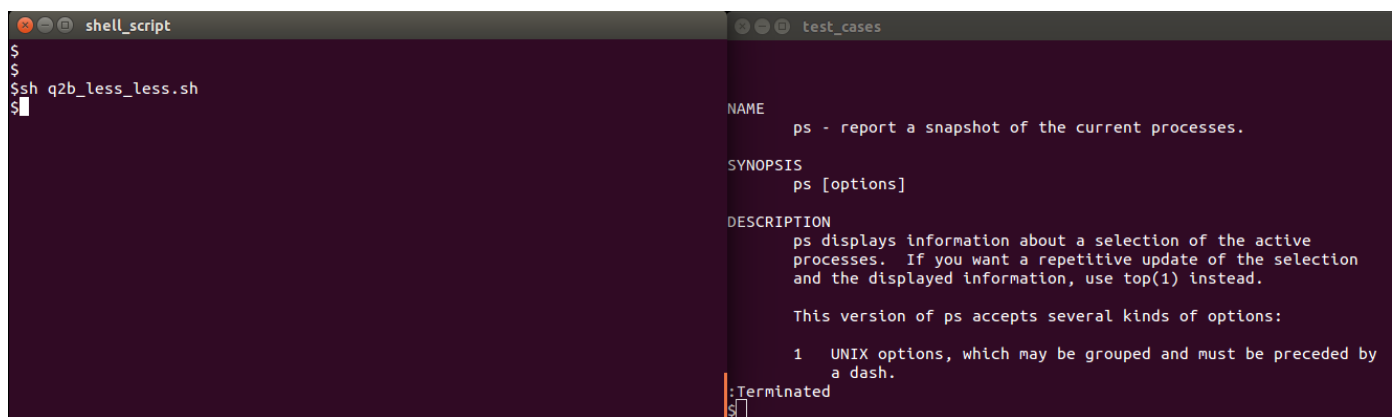
Here is an example session from each of these two terminals. The “test_cases” terminal is about to be used for some casual browsing of the man pages:



```
shell_script
$
$
$
$

test_cases
$
$
$man ps | less
```

However, the script developed in this part is able to cut that short when we run it in the “shell_script” window (who needs man pages, right?)



```
shell_script
$
$
$sh q2b_less_less.sh
$

test_cases
NAME
    ps - report a snapshot of the current processes.

SYNOPSIS
    ps [options]

DESCRIPTION
    ps displays information about a selection of the active
    processes.  If you want a repetitive update of the selection
    and the displayed information, use top(1) instead.

    This version of ps accepts several kinds of options:

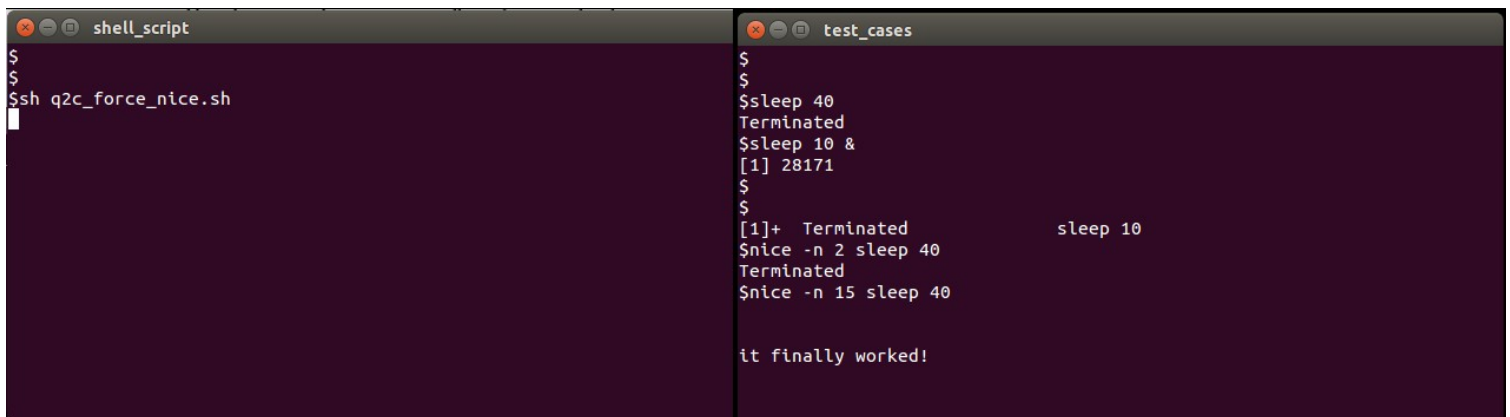
    1  UNIX options, which may be grouped and must be preceded by
        a dash.
:Terminated
$
```

Part c) (30 marks) The “nice” command is an important tool to assign priority to process on systems with many users. When a system becomes crowded with many users, it could be desirable to FORCE users to run some of their programs with increased niceness. In fact, let's do just that:

Write a shell script called “**q2c_force_nice.sh**” with the following specifications:

- When executed, causes any currently running processes of the “sleep” command started by your user account to be terminated IF it has a niceness value of 4 or lower.
- Does NOT terminate any “sleep” command with niceness of 5 or higher.
- The script itself runs forever unless explicitly killed (e.g., with ctrl-c).
- Any new “sleep” jobs run by your user account continue to be terminated, for as long as the script is running, again following the niceness rule (≤ 4 are terminated, >4 not terminated).
- Does NOT terminate any other process, including processes that you own and especially not processes owned by other users. This point is regardless of niceness level.

This code can be tested in the same fashion (using 2 terminals) as was used for part b). Here is a sample session:



```
shell_script
$
$
$ ssh q2c_force_nice.sh

test_cases
$
$
$ sleep 40
Terminated
$ sleep 10 &
[1] 28171
$
$
[1]+  Terminated                  sleep 10
$ nice -n 2 sleep 40
Terminated
$ nice -n 15 sleep 40

it finally worked!
```

3. Managing Files (a.k.a. A System to Organize your Vacation Photos)

(30 marks)

When files are spread across multiple directories, we can use shell scripts to re-organize and sort them in many ways. In this question we'll gather image files from many folders based on their date. The **Problem3** folder in the zip file holds 2 examples for you to test this on. The first is called **MontrealTest**, which has some photos taken by the instructors and TAs over the past year. We stored our images in a variety of folders, and we'd like your script to find and sort them by their modification date, to help us turn our disorganized directories into a seasonally-relevant output like this one:

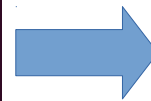
```
$ls -lR Problem3/MontrealTest/
Problem3/MontrealTest/:
total 12
drwxrwxr-x 2 dmegeer nogroup 4 Sep 22 18:02 daves_images
drwxrwxr-x 2 dmegeer nogroup 3 Sep 22 18:13 gregs_photos
drwxrwxr-x 2 dmegeer nogroup 5 Sep 22 17:40 photos_by_harth
drwxrwxr-x 2 dmegeer nogroup 4 Sep 22 18:01 sandeeps_collection

Problem3/MontrealTest/daves_images:
total 906
-rw-r----- 1 dmegeer nogroup 280586 Sep 22 17:58 mtl10.jpg
-rw-r----- 1 dmegeer nogroup 455437 Sep 22 17:58 mtl7.jpg

Problem3/MontrealTest/gregs_photos:
total 389
-rw-r----- 1 dmegeer nogroup 307991 Sep 22 18:03 mtl1.jpg

Problem3/MontrealTest/photos_by_harth:
total 1295
-rw-r----- 1 dmegeer nogroup 437881 Sep 22 18:03 mtl11.jpg
-rw-r----- 1 dmegeer nogroup 376483 Sep 22 17:57 mtl5.jpg
-rw-r----- 1 dmegeer nogroup 272425 Sep 22 18:01 mtl9.jpg

Problem3/MontrealTest/sandeeps_collection:
total 778
-rw-r----- 1 dmegeer nogroup 364466 Sep 22 18:00 mtl4.jpg
-rw-r----- 1 dmegeer nogroup 382957 Sep 22 18:02 mtl8.jpg
$
```



The image-strip above was produced using a convenient tool known as “convert” that comes from the “ImageMagick” software package. This is not a default package, so you may need to install it if you're working from home. It is available on the SOCS machines and the Trottier labs. To build an image-strip, you can use the “-append” flag. The input image names are given as an ordered argument list, with the first image that you specify and then to specify the resulting output filename.

E.g., `$ convert -append fall.jpg winter.jpg spring.jpg summer.jpg merged_seasons.jpg`

To install this program on your local Ubuntu, the command is “`sudo apt-get install imagemagick`”.

Write a shell script called “**q3_image_sorter.sh**” with the following specification:

- Accepts the name of 1 directory as a command-line argument. Produces an image-strip that includes all “jpg” images found in that folder or any of its descendant sub-folders, no matter of their depth. The image-strip must present the images sorted by their **modification time**, with the oldest file appearing at the top of the image and the newest at the bottom.
- If your script is called with an incorrect (or missing) command-line argument, it must print a reasonable error message to the screen and exit.
- The image-strip output should be produced as a newly created image file in the working directory (that is the one where your script has been called from), with extension “jpg”. This output image should have a filename that is identical to the argument given to your script, but with each “/” in the directory path replaced by an under-score:
 - e.g., “`$ sh q3_image_sorter.sh Problem3/SimpleTest`” leads to an image with name “Problem3_SimpleTest.jpg”. This is actually a good test to start with, it should produce the image below, made up of simple images of numbers. If they count upwards, your script works correctly.

1

2

3

4

Submitting your solutions:

Please make sure that you've run all of the tests mentioned in this document, plus some more that you can think of for each script. Ensure that your code runs on mimi or the Trottier machines (they are identical, so either is fine).

Submit a single zip file to My Courses called “**A1_solutions.zip**” that contains 4 files:

1. **A1_written_solutions.txt**
2. **q2b_less_less.sh**
3. **q2c_force_nice.sh**
4. **q3_image_sorter.sh**

You can create this file with the command:

```
$ zip A1_solutions.zip A1_written_solutions.txt q2b_less_less.sh q2c_force_nice.sh q3_image_sorter.sh
```

The submission deadline is 1 minute before midnight on Sunday, October 4th.