Assignment 3

CSCI 2132: Software Development

Due Febuary 25, 2019

General Instructions: How to Submit Your Work

You must submit your assignment answers electronically:

- Change into your subversion directory on bluenose: `cd ~/csci2132/svn/CSID`.
- Create a directory to hold your assignment answers: `mkdir a3`.
- Change into your assignment directory: `cd a3`.
- Create files inside the a3 directory as instructed in the questions below and put them under Subversion control using `svn add <filename>`. **Only add the files you are asked to add!**
- Once you are done answering all questions in the assignment (or the ones that you are able to answer—hopefully all), the contents of your a3 directory should look like this:

```
a2
├─ a3q1.txt
├─ a3q2.log
├─ file.txt
├─ compress_large_files.sh
└─ view.sh
```

Submit your work using `svn commit -m"Submit Assignment 3"`.

(Q1) Theory: Soft links, hard links, and inodes (12 marks)

Answer each of the following questions. Collect your answers in a plain text file `a3q1.txt` and put this file under Subversion control using `svn add a3q1.txt`. The answer to each part of this question should be marked with the part it belongs to, like this:

```
a3q1.txt
(a) A fox has red fur and sharp teeth.
...

(b) It said to install Windows 7 or better, so I installed Linux.
...


...
```

(a) Explain the difference between a job and a process.

(b) Explain the difference between a foreground process and a background process.

(c) Explain the difference between a centralized version control system and a distributed version control system.

(d) Clearly explain the steps you need to take to suspend a job that was started from your shell and then resume it.

(e) Clearly explain how to start a program from your shell as a background process (without starting it as a foreground process first).

(f) Clearly explain how to switch a foreground process to the background from your shell.

(Q2) Practice: Capturing program output and extended regular expressions (10 marks)

To answer this question, record the sequence of commands you perform using the `script` command as in Question 2 of the first assignment. To review how to use this command, refer to Question 2 of the first assignment or run `man script` on `bluenose`. Record your sequence of commands and their outputs in a file `a3q2.log`. Question (b) asks you to create a file `file.txt`. Put both `a3q2.txt` and `file.txt` under subversion control.

(a) Use the commands `ls` and `which` (and no other commands) to find out the owner, group, and size of the `less` program. (Be sure to run this on `bluenose` because it may have different sizes even on two different Linux systems.) You may need to consult the manpage for the `which` command to learn what it does. Your solution should consist of a single command line, that is, you should not enter one command, press `Enter`, and then enter another command and press `Enter` again; everything should be typed after the same shell prompt and you should press `Enter` only once. You need to find a way to pass the output of `which` to `ls` as a command line argument.

(b) Create a file `file.txt` with anywhere between 10 and 20 lines in it. Now provide a single command line that uses the commands `cat`, `wc`, and `head` to display the first half of this file. For example, if `file.txt` has 15 lines, you should display the first 7 lines. If it has 16 lines, you should display the first 8 lines. Since you created the file, you know how many lines it has. Thus, if `file.txt` has 15 lines, you could simply write `head -n 7 file.txt`. Do not do this. Instead, provide a command line that works correctly no matter how many lines there are in `file.txt`. You command line will likely consist of multiple commands separated by "`;`", one that assigns the length of `file.txt` to a variable, one that divides this variable by two, and one that uses this variable to instruct `head` how many lines to display.

(c) Use `egrep` (or `grep -E`) to find all words in the file `/usr/share/dict/linux.words` on `bluenose` that contain the same 3-letter sequence at least three times. For example, one such word matched by the correct regular expression is "cha-cha-cha". It contains the 3-letter sequence "cha" three times.

(d) The example of a regular expression for an IP address I gave in class was too permissive because it only checked that each of the four numbers has 1–3 digits. A valid IP-address is of the form $a.b.c.d$, where $0 \le a, b, c, d \le 255$. Provide a regular expression that checks not only that each number is 1–3 digits long but that each number is between 0 and 255.

(Q3) Practice: Shell scripting (10 marks)

`gzip` is a program that can be used to compress files in order to either save disk space or be able to send these files across a slow network connection more efficiently. Given a file with name `file`, it produces an output file `file.gz` that is a compressed representation of the contents of `file`. The inverse of `gzip` is `gunzip`. Given a file `file.gz` produced by `gzip`, it restores the uncompressed file `file`. To get comfortable with these two tools, it may be helpful that you use some text file, compress it using `gzip`, inspect its contents using `cat` (which will look like random junk), and then uncompress it again using `gunzip`. It may also be useful to look at the sizes of the original file, the compressed file, and the restored file, to convince yourself that `gzip` does indeed decrease the size of the file (at least if the input is a reasonably large text file) and `gunzip` restores the original file exactly. (Note that these steps are not part of the assignment; they are meant to increase your comfort with using `gzip` and `gunzip`.)

For this question, your goal is to store large files in compressed form while not compressing small files. Part (a) of this questions asks you to write a shell script `compress_large_files.sh` that accomplishes this. Part (b) asks you to complement this compression script with a utility `view.sh` that displays the contents of a given file. However, the argument of `view.sh` may be a compressed file or an uncompressed file. `view.sh` must handle both types of files correctly and display the uncompressed text in the file in both cases.

(a) Write a script `compress_large_files.sh`. This script accepts one or more command line arguments. The first argument has to be an integer; let's call it *size*. If this is the only command line argument, `compress_large_files.sh` inspects all files in the current working directory and compresses every file of size at least *size*.

If there is more than one command line argument, all arguments except the first one must be valid directories. In this case, `compress_large_files.sh` inspects the files in each of these directories and, once again, compresses every file of size at least *size*.

If successful, your script should not produce any output on `stdout`. You script should produce the following error messages:

- If the wrong number of command line arguments is given, a usage message `USAGE: compress_large_files.sh size [dir ...]` should be printed.
- If the first argument is not a number, a message `ERROR: xxx is not a number` should be printed, where `xxx` is the argument provided by the user.
- If any argument other than the first one is not a valid directory, a message `ERROR: xxx is not a directory` should be printed, where `xxx` is the argument in question.
- If a directory to be searched for files or a file in such a directory cannot be read, a message `ERROR: Cannot read file xxx` should be printed, where `xxx` is the file or directory in question. Subdirectories of the current working directory or of the directories provided on the command line should not be search recursively and should be ignored silently.

(b) Write a script `view.sh` that takes one or more command line arguments. Each argument must be the name of a readable file. For each argument, `view.sh` should check whether this is a file that was compressed using `gzip`. If so, it should uncompress it using `gunzip` and display the result using `cat`. (You may also explore whether `gunzip` can display its output directly, without using `cat` as a helper.) If the file was not compressed using `gzip`, the file should be displayed without decompressing it. There are two important requirements for `view.sh`:

- While the default behaviour of `gunzip` is to delete the `file.gz` file it is given and create the uncompressed file `file` from it, `view.sh` is not allowed to do this. There are many ways to achieve this. One of them is to copy the file to be decompressed into the `/tmp` directory, decompress it there, display it, and delete it from `/tmp`.
- Second, `view.sh` should display only the contents of the files that were passed to it. For each file that cannot be read, it should print an error message `ERROR: Cannot read file xxx`, where `xxx` is the file in question. No other output must be produced.

Most of this question should be straightforward. There are two parts that require some creativity:

- How to check whether a file is compressed? `gunzip` can help you with that. Just try to run it on the given file. If the file is a gzipped file, this just works. Otherwise, it fails and `gunzip` lets you know via its exit code.
- The previous suggestion leads to another wrinkle: when `gunzip` fails, it happily informs us with an error message, but `view.sh` is not allowed to produce any output other than its own error messages. You need to figure out how to silence `gunzip`. `man gunzip` and a careful read of that manpage and a Google search for `/dev/null` lead you to two possible ways to do this.