CS 97 Final Fall '20

1. The Liddil test checks whether the strings A and B are known to name the same file. It works this way:

- Run the shell command `ls -Liddil` with the two operands A and B.
- If the command fails, the Liddil test fails.
- Otherwise, compare the two lines of output. The strings A and B name the same file if and only if the two lines are the same, except for the occurrences of 'A' and 'B' at the respective ends.

This test assumes you are running on SEASnet, are testing files in your home directory, and .snapshot directories are not involved; you can continue to make these assumptions in the Liddil test questions.

Simplify the description of the Liddil test as best you can, while staying within the constraints of using the `ls` command to implement it. Describe why each simplification works, and explain why other plausible simplifications would not work.

---

2. Write a shell command named `Liddil-test` that quietly succeeds if given two operands for which the Liddil test succeeds, quietly fails if given two operands for which the Liddil test fails, and fails with a diagnostic otherwise. For example, on SEASnet, `Liddil-test /bin/sh /usr/bin/bash` should succeed, whereas `Liddil-test '' /` should quietly fail because the empty string does not name a file. When given two operands your command should not generate any output: it should merely succeed or fail, so that the user can write a shell command like this:

```
if Liddil-test /bin/sh /usr/bin/bash; then
  echo 'same file'
else
  echo 'differing files'
fi
```

without getting any stray messages. Your command should not take any options, and should treat all operands as file names even if they begin with '-'; for example, `Liddil-test -E .` should quietly fail unless you happen to have a symbolic link like `-E -> ..`

---

3. Suppose some other program is modifying the file system at the same time that your Liddil-test program runs. How would this affect the validity of your program's results? Briefly explain.

---

4. In .git/objects Git splits the 40-character SHA-1 text checksum into two parts, of length 2 and 38 characters respectively. Why doesn't Git instead split them into equal parts of 20 characters each? Wouldn't that be fairer or more efficient in some way? Or, of 2 vs 38 is good, why wouldn't 1 vs 39 (or 3 vs 37, etc.) be better? Briefly explain.

---

5. Data backup systems use deduplication, compression, and encryption heavily. Compare and contrast their use of these three technologies to how Git uses them. Assume Git plumbing only; do not worry about porcelain or add-ons.

---

6. Suppose you wanted to add checkpoint/restart capability to a working randall program running on SEASnet. The idea is that you run the program like this:

```
randall 100000000000 >output
```

and that if the system crashes while in the middle of a run, after it reboots you can continue where you left off by doing this:

```
randall 100000000000 >>output
```

Explain how to modify your randall implementation to support checkpoint/restart in this way by using functions like `fseek`, `ftell`, or `lseek`. Use explicit code snippets in your explanation where appropriate. Explain what trouble your modifications would have when doing checkpoint/restart if the modified randall outputs to a pipe to some other program, instead of outputting to regular file.

---

7. Compare and contrast the following approaches to detecting and/or preventing integer overflow errors in your C or C++ programs:

   run the program under GDB

   ```
   gcc -Wall -Wextra
   gcc -fsanitize=undefined
   ```

   rewrite your program in: Emacs Lisp JavaScript Python

---

8. Consider the following potential criticisms of Node.js:

- It doesn't support multithreaded applications, which makes many apps hard to scale.
- It's tied closely to Google's V8 JavaScript engine, and so is not portable to other platforms.

- It's too low level; if you want to build a real app you must write everything nearly from scratch, or pull in other peoples' code like crazy.
- JSX is too complicated and its learning curve is too steep, compared to other approaches.

Add one other reasonable criticism to this list.

For each of the five criticisms (the above four, plus your fifth):

- Argue that the criticism is a reasonable one, as best you can.
- Give a defense of Node.js against the criticism, as best you can.
- Explain how well the criticism applies to the project that you worked on for this course.

---

9. The man page for `git merge` has the following words of wisdom:

Running git merge with non-trivial uncommitted changes is discouraged: while possible, it may leave you in a state that is hard to back out of in the case of a conflict.

Write a shell script that illustrates this advice. Your shell scrimpt should set up a Git repository and working files from scratch (no fair cloning from any other source), and run `git merge` with nontrivial uncomitted changes. Explain the resulting state, and why this sort of state is hard to back out of in the more-typical case where you don't have a complete shell script that you can replay.

---

10. The man page for `git rebase` does not have words of wisdom that are like those mentioned in the previous (`git merge`) question. Should it? Briefly explain.

---

11. Suppose we want to reimplement Emacs using Node.js and React, instead of the existing Emacs code base that uses C as its core. We want existing Emacs scripts and and keystrokes to work unchanged, so that current users can continue to get their work done.

Does this idea make sense? Briefly explain why or why not. Either way, give the biggest obstacles you see to making it work.

---

12. Assuming you could pull off the idea of reimplementing Emacs in Node.js and React, how does it compare to the existing practice of using Emacs as a development environment for Node.js and React applications? Would

there be significant advantages to the proposed implementation compared to the current one? Briefly explain.