UCLA Computer Science 97 Midterm 1 Solutions

Spring 2020

These solutions are intended to be as comprehensive and instructive as we can make them. They are not exemplars of the level of detail expected of you in the actual midterm.

1. **What is the likely typo in the following shell command, and why is it such a serious typo?**

```
rm *.[Oo][Uu][Tt] * .o *.a
```

Answer: The typo is that there is a space between `*` and `.o`. This is a serious typo because instead of removing `*.o`, which means all files ending in `.o`, we are going to remove `*`, which means all files in the current directory.

It should be noted that this notation is **not** (although it is very similar to) regular expression syntax. The asterisks and square brackets are known as wildcards. The asterisk `*` can represent any number of characters. It is equivalent to the meaning of `.*` in regular expressions.

2. **A set of read/write/execute permissions on a file is called "sensible" if the owner has all the permissions of the group, and the group has all the permissions of others. For example, 551 (octal) is sensible, whereas 467 (octal) is not.**

   a. **Briefly explain why non-sensible permissions don't make much sense.**

   Answer: In Unix terminology, the *owner* is a single person who is considered the primary owner of the file. The *group* is generally a group of people who are connected to the file in some way (but not as much as the *owner*).

   A non-sensible permission essentially means either of the following:

   — The *owner* – the person who generally needs the most access – has fewer permissions than a *group* of people typically less connected with the file.
   — Anyone else on the system is permitted to use the file in more ways than even the *group* of people considered highly connected to the file.

   Neither scenario is reasonable.

   b. **How many distinct sensible permissions are there? Explain.**

   Answer: The possible permission bits for the *group* are constrained by the

permission bits for *others*, and similarly the bits for the *owner* are constrained by the *group*. If a 1 appears for a certain permission in *others*, then it must also be set to 1 for that same permission in the *group* and *owner* bits.

From here, we can enumerate the possible cases:

If *other*'s bits all 0 (**1** case, 000):

Possible group bits	000	One bit set (3 cases)	Two bits set (3 cases)	111
# of possible owner bits	8	4	2	1
	8	12	6	1
			Total:	27 × 1 = 27

If *other* has one bit set (**3** cases, 001, 010, 100):

Possible group bits	Neither of the remaining 2 bits are set	One of the remaining bits is set (2 cases)	All remaining bits are set
# of possible owner bits	4	2	1
	4	4	1
		Total:	9 × 3 = 27

If *other* has two bits set (**3** cases, 011, 101, 110):

Possible group bits	The remaining bit is not set	The remaining bit is set
# of possible owner bits	2	1
	2	1
	Total:	3 × 3 = 9

If *other*'s bits all 1 (1 case, 111):
In this case, the only sensible permissions are when owner's bits = group's bits = 111. **Total:** 1 × 1 = 1.

Therefore, the total number of distinct sensible permissions is:

$$27 + 27 + 9 + 1 = \mathbf{64}.$$

3. **Give two good reasons why backups do not suffice for version control.**

Answer: Backups in general are intended for a different use case (disaster recovery) than version control (collaboration). There are many specific reasons that all allude to this theme, some of which we have recorded below:

— Version control is not just about the past/history, it's also about planning for the future. Often, possible future changes to a program are not compatible (i.e. moving forward with using one library vs. another). Version control systems include tools to make this process easier, whereas backups do not.

— Backups are generally intended to be complete copies of the resource. This is inappropriate for version control, which often deals with a rapidly changing resource (e.g., a project being actively worked on) where techniques like delta encoding and compression can drastically reduce the space and time needed to store new snapshots.

— It is typically cumbersome to share backups (which should ideally be stored on dedicated storage devices, including some unusual ones) with other people. Dedicated version control systems (including Git, Subversion, Mercurial, Bazaar, CVS, etc.) typically provide a way of sharing and collaborating through the Internet.

4. **Give an example of how renaming a dangling symbolic link can transform it into a non-dangling symbolic link.**

Answer: There are two possible solutions, given the ambiguity of the term "renaming". The most important part is recognizing that symbolic links point to file *names* rather than files themselves.

The first solution is using the `mv` command to move (i.e. "rename" the absolute path of) the symbolic link from its current directory to another that does have the file being linked to:

```
$ mkdir temp
$ echo OK > temp/b
$ ln -s b c          # Note that c is a dangling link, as the file b does not exist
$ cat c              # in the current directory.
cat: c: No such file or directory
$ mv c temp
$ cat temp/c         # Now temp/c is no longer dangling, as temp/b exists.
OK
```

The second solution is to have the symbolic link point to itself:

```
$ ln -s a b
$ cat b              # Note that b is a dangling link to the nonexistent a.
cat: b: No such file or directory
$ mv b a
$ cat a              # Note that a is now a link to itself.
cat: a: Too many levels of symbolic links
```

5. **Explain how to arrange for Emacs to treat `C-t` (i.e., control T) as a command that causes Emacs to issue a message like this:**

    ```
    It is now Tue Apr 27 10:30:34 2020.
    ```

    in the echo area. The message should contain the current date and time.

    Answer: Here's an outline on how one could do this. The first piece that is essential is how to bind the `C-t` key combination to a command. As covered in lecture, this could be accomplished with the `global-set-key` command. The second piece is making a command that formats the time. `current-time-string` mentioned in lecture and `concat` would work for this, as well as `format-time-string`. The last piece is to issue a message in the echo area; again as mentioned in lecture, `message` is the function we would want to use here.

    If you were able to get as far as this, then you should probably get a lot of credit for the problem already. What follows is a full solution to accomplish this.

    We first define the following Emacs Lisp function by typing it in the `*scratch*` buffer and evaluating the code with `C-j`:

    ```
    (defun print-time ()
      (interactive)
      (message (concat "It is now " (current-time-string) ".")))
    ```

    Equivalently, we could use `format-time-string`:

    ```
    (defun print-time ()
      (interactive)
      (message (format-time-string "It is now %a %b %e %T %Y.")))
    ```

    Note that the `interactive` special form turns the Lisp function into a command.

Once this is done, we create the `C-t` key binding, by executing the following

```
M-x global-set-key <RET> C-t print-time <RET>
```

This binds the `C-t` key combination to the `print-time` command we just defined.

6. **If a Python class *C* has a method *M* and a class variable *V*, what sort of thing does `C.__dict__.items()` return and why?**

   Answer: To answer this question, we need only try it out in a Python interpreter:

```
$ python3
Python 3.8.2 (default, Apr  8 2020, 14:31:25)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> class C:
...   V = 'cs97'
...   def M():
...     pass
...
>>> C.__dict__.items()
dict_items([('__module__', '__main__'),
            ('V', 'cs97'),
            ('M', <function C.M at 0x7f097990c4c0>),
            ('__dict__', <attribute '__dict__' of 'C' objects>),
            ('__weakref__', <attribute '__weakref__' of 'C' objects>),
            ('__doc__', None)])
```

   (Some pretty-printing is done here.)

   We see that *C*`.__dict__.items()` returns a `dict_items` object containing (among others) the entries

```
'V' → the value we set the class variable V to
'M' → <the function C.M>
```

   This is because the `__dict__` property on objects contains all of an object's writable attributes, which includes both its data members and methods.

7. **By default, the Python expression '`sys.modules`' signals a `NameError`, but if you execute '`import sys`' first the expression does not signal that error.**

   a. **Briefly explain why not.**

Answer: By default, the Python interpreter only binds a small set of identifiers (variable names) to values, which evidently excludes the identifier 'sys'. Put simply, there is no variable named 'sys' that exists by default. This is presumably done to improve the start-up speed of the Python interpreter, and to prevent conflicts for scripts that want to assign some other values to the identifier 'sys'.

Once the `import` statement is executed, however, the `sys` module is imported and bound to the identifier 'sys'.

b. Give an example of using the value of the expression '`sys.modules`' to find out something about your Python process. Your example should explore at least one level past the value of `sys.modules` itself.

Answer: Many examples are possible here. Consider the following Python interactive session transcript:

```
$ python3
Python 3.8.2 (default, Apr  8 2020, 14:31:25)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.modules.keys()
dict_keys(['sys', 'builtins', '_frozen_importlib', '_imp', '_warnings',
           '_frozen_importlib_external', '_io', 'marshal', 'posix',
           '_thread', '_weakref', 'time', 'zipimport', '_codecs',
           'codecs', 'encodings.aliases', 'encodings', 'encodings.utf_8',
           '_signal', '__main__', 'encodings.latin_1', '_abc', 'abc',
           'io', '_stat', 'stat', '_collections_abc', 'genericpath',
           'posixpath', 'os.path', 'os', '_sitebuiltins', '_locale',
           '_bootlocale', 'site', 'readline', 'atexit', 'rlcompleter'])
```

Here, we were able to find a list of all imported modules in the current Python process through `sys.modules.keys()`.

8. Consider the following shell script.

```
#!/bin/sh
atom='[a-zA-Z0-9]+'
string='\\"([^"\]|\\.)*\\"'
word="($atom|$string)"
words="$word(\\.$word)*"
```

```
grep -E "$words" | grep ' '
```

a. Explain briefly what this shell script does, from its user's viewpoint.

Answer: This shell script prints any line in the standard input that satisfy *both* of the following:

— Contains a space.
— Contains at least one of the following:
(1) an alphanumeric character (`$atom`), or
(2) a (possibly empty) sequence of non-'"' (literal double-quote) characters that is surrounded by '\"' (literal backslash followed by literal double-quote). If '\' (literal backslash) appears within the string, then it must start a well-formed escape sequence (it has another character that follows it) before the closing '\"' (`$string`).

Note that the '`(\\.$word)*`' part of `$words` does not change the set of lines matched, since having something in a line match `$word` is both necessary and sufficient for a line to match `$words`.

Note additionally that the '`\\`' in `$words` becomes a single backslash by the time it reaches `grep` since '"' (a double quote) is used, but the '`\\`' in `$string` remains '`\\`' by the time it reaches `grep` since ''' (a single quote) is used.

Examples of input lines that would be output by the shell script verbatim:

`1 2`	Has a space and matches criterion (1)
`hello, world`	Has a space and matches criterion (1)
`\" hello`	Has a space and matches criterion (1)
`\" \"`	Has a space and matches criterion (2)
`\"| \"`	Has a space and matches criterion (2)
`\"Hello, world!\n\"`	Has a space and matches both criteria

Examples of input lines that would be gobbled (not output) by the shell script:

`123`	Does not have a space
`hello`	Does not have a space
`": ]"`	Does not match criterion (2) as `\"` is not used
`\"" \"`	Does not match criterion (2) as `"` appears between `\"`

`\" \\"` Does not match criterion (2) as \ is followed by \"

b. How would the script's behavior change if you removed the '-E' from this script?

Answer: If we were to remove the '-E' flag, then POSIX Basic Regular Expression (BRE) is used instead of the Extended Regular Expression (ERE). This means that many meta-characters used, including the '(', '|', and ')' in `$word`, are treated as normal characters to match rather than the start of group, disjunction, and end of a group, respectively.

c. Modify the original script so that it calls grep just once instead of twice, without changing the script's I/O behavior.

Answer: Replace the last line with

```
grep -E "$words" | sed -n '/ /p'
```

The second command does the same thing as the command "`grep ' '`", but without using grep. (Other answers are possible too.)

9. Consider the following shell transcript:

```
$ git clone https://github.com/git/git.git
Cloning into 'git'...
remote: Enumerating objects: 274, done.
remote: Counting objects: 100% (274/274), done.
remote: Compressing objects: 100% (162/162), done.
remote: Total 286088 (delta 139), reused 215 (delta 112), pack-reused 285814
Receiving objects: 100% (286088/286088), 135.55 MiB | 27.26 MiB/s, done.
Resolving deltas: 100% (212818/212818), done.
```

a. Say briefly what this command does, and why you might want to run it.

Answer: This command creates a clone of the Git repository containing the Git source code on your local machine. You may want to run it in order to look at and experiment with the Git source code (e.g., contribute to the code, debug, etc.).

b. How can it make sense to use Git to get Git's source code, when one needs to start with Git's source code in order to build Git in the first place? Explain why this circular process isn't a fundamental roadblock.

Answer: This circular process is not a fundamental roadblock because Git is

often distributed as a pre-compiled binary (i.e. program) for your platform. Thus, we do not need to start with the source code to use Git, since we can simply download and use the executables.

Even if your platform does not yet have pre-compiled binaries, you can download the Git source code as a zip file or tarball from GitHub or kernel.org, thus obviating the need for a working Git installation to build Git.

c. Consider the following continuation of the shell transcript:

```
$ cd git
$ git log | tail -n 5
commit e83c5163316f89bfbde7d9ab23ca2e25604af290
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date:   Thu Apr 7 15:13:13 2005 -0700

    Initial revision of "git", the information manager from hell
```

Git's first version v0.99 wasn't created until July 10, 2005, so how can this Git commit possibly have been created on April 7 of the same year? Briefly explain.

Answer: There are two possible solutions we could think of:

Firstly, the date of any commit can be set to an arbitrary date by using the "`--date`" option for "`git commit`".

Additionally, it is possible that Torvalds had an experimental and unreleased version of Git already available by April 7, 2005.

N.B.: Both of the above are valid answers, but only the latter is true in real life.