Assignment 3. Modifying and rewriting software

Useful pointers

- Michael Johnson, Diff, Patch, and Friends, Linux Journal 28 (1996-08)
- David MacKenzie, Paul Eggert, and Richard Stallman, Comparing and merging files, version 3.7 (2018-01-07)
- The Python Tutorial (2019-10-09)

Laboratory: Installing a small change to a big package

Keep a log in the file `lab3.txt` of what you do in the lab so that you can reproduce the results later. This should not merely be a transcript of what you typed: it should be more like a true lab notebook, in which you briefly note down what you did and what happened.

You're helping to build an application containing a shell script that invokes the `ls` command to get file status. Your application is running atop the Maroon Chapeau Enterprise Linux 8 distribution, which uses the `ls` implementation supplied by Coreutils 8.29. You've been running into the problem that some users create a shell script `la` with the following contents:

```
#!/bin/sh
exec ls -a "$@"
```

For these users the command `la -A` is therefore equivalent to `ls -a -A`. Unfortunately, with Coreutils `ls`, the `-a` option always overrides the `-A` option regardless of which option is given first, so the `-A` option has no effect in `la`. For example, if the current directory has two files named `.foo` and `bar`, the command `la -A` outputs four lines, one each for `.`, `..`, `.foo`, and `bar`. These users want `la -A` to output just two lines instead, one for `.foo` and one for `bar`. That is, for `ls` they want a later `-A` option to override any earlier `-a` option, and vice versa.

You've been asked to look into the problem and fix it. Start by verifying that the problem still occurs in `/usr/bin/ls` on the SEASnet GNU/Linux servers. Also, check the Coreutils version that `/usr/bin/ls` corresponds to.

You discover that the problem is a known bug with Coreutils, Bug#30963. The bug report has a patch intended for publication in a later Coreutils release; see its Message #10. You don't want to wait for the later release to be installed on your system, so you decide to build a copy of Coreutils 8.29 with just this patch added, as follows:

1. Grab the Coreutils 8.29 source code compressed tarball and verify its signature with the GNU keyring by running the shell command `gpg --verify --keyring ./gnu-keyring.gpg coreutils-8.29.tar.xz.sig` in your directory. Note any problems with this verification, and briefly explain why they happen.
2. Compile and install your copy of Coreutils into a temporary directory of your own. Note any problems you run into.
3. Reproduce the bug on your machine with the unmodified version of coreutils. The test case that comes with the patch should give you ideas for how to reproduce the bug.
4. Apply the patch of Bug#30963 Message #10. Note any problems that occur (they should not prevent the modified coreutils from working).
5. Type the command `make` at the top level of your source tree, so that you build the fixed version, without installing it into your temporary directory. For each command that gets executed, explain why it needed to be executed (or say that it wasn't neeeded).

6. Make sure your change fixes the bug, by testing that the modified `ls` works on the same tests where the unmodified `ls` didn't work.

Q1. Does the patch improve the performance of `ls` or make it worse? Briefly explain.

Q2. If your company adopts this patched version of Coreutils instead of the default one, what else should you watch out for? Might this new version of Coreutils introduce other problems with your application?

Homework: Rewriting a script

Consider the old-fashioned Python 2 script `randline.py`.

Q3. What happens when this script is invoked on an empty file like `/dev/null`, and why?

Q4. What happens when this script is invoked with Python 3 rather than Python 2, and why? (You can run Python 3 on the SEASnet hosts by using the command `python3` instead of `python`.)

Write a new script `shuf.py` in the style of `randline.py` but using Python 3 instead. Your script should implement the GNU `shuf` command that is part of GNU Coreutils. You should already have a copy of the `shuf` source code and documentation; look at the files you created in the laboratory. However, GNU `shuf` is written in C, whereas you want a Python implementation so that you can more easily add new features to it.

Your program should support the following `shuf` options, with the same behavior as GNU `shuf`: `--input-range` (`-i`), `--head-count` (`-n`), `--repeat` (`-r`), and `--help`. As with GNU `shuf`, if `--repeat` (`-r`) is used without `--head-count` (`-n`), your program should run forever. Your program should also support zero non-option arguments or a single non-option argument "`-`" (either of which means read from standard input), or a single non-option argument other than "`-`" (which specifies the input file name). Your program need not support the other options of GNU `shuf`. As with GNU `shuf`, your program should report an error if given invalid arguments.

If you have trouble with `optparse` under Python 3, you can use the `argparse` module instead. Your `shuf.py` program should not import any modules other than `argparse`, `string` and the modules that `randline.py` already imports. Don't forget to change its usage message to accurately describe the modified behavior.

Q5. What happens when your `shuf.py` script is invoked with Python 2 rather than Python 3, and why?

Submit

Submit the following files.

- The file `lab3.txt` as described in the lab.
- A file `hw3.txt` containing the answer to questions Q1 through Q5 noted above.
- The file `shuf.py` as described in the homework.

All files should be ASCII text files, with no carriage returns, and with no more than 80 columns per line. The shell command:

```
expand lab3.txt hw3.txt | awk '/\r/ || 80 < length'
```

should output nothing.

© 2005, 2007–2013, 2015, 2018, 2019 Paul Eggert. See copying rules.
$Id: assign3.html,v 1.33 2019/10/14 19:00:13 eggert Exp $