Assignment 1. Getting to know your system

Laboratory: Linux and Emacs scavenger hunt

Instructions: Do this assignment on the SEASnet GNU/Linux servers `lnxsrv06`, `lnxsrv07`, `lnxsrv09`, or `lnxsrv10`, with `/usr/local/cs/bin` prepended to your `PATH`. Use the commands that you learned in class to find answers to these questions. Don't use a search engine like Google, and don't ask your neighbor, don't use GitHub, etc. If you need a hint, ask the TA. When you find a new command, run it so you can see exactly how it works. In addition to turning in the answers to these questions, turn in a description of your session discovering them.

As you do actions, use Emacs to record your keystrokes and each answer in plain-text files `key1.txt` and `ans1.txt` that you will submit as part of the assignment. If you prefer, you can record and submit Org files `key1.org` and `ans1.org` instead of `.txt` files; see Hello,Worg for Org introductions and tutorials.

1. What shell command uses the `man` program to print all the commands that have a specific word in their man page (or at least the description part of the man page)? (hint: `man man`)
2. Where are the `mv` and `sh` programs located in the file system? List any shell commands you used to answer this question.
3. What executable programs in `/usr/bin` have names that are exactly two characters long and end in `r`, and what do they do? List any shell commands you used to answer this question.
4. When you execute the command named by the symbolic link `/usr/bin/emacs`, which file actually is executed? List any shell commands you used to answer this question.
5. What is the version number of the `/usr/bin/gcc` program? of the plain `gcc` program? Why are they different programs?
6. The `chmod` program changes permissions on a file. What does the symbolic mode `u+sx,o-w` mean, in terms of permissions?
7. Use the `find` command to find all directories modified in the last four weeks that are located under (or are the same as) the directory `/usr/local/cs`. List any shell commands you used to answer this question.
8. Of the files in the same directory as `find`, how many of them are symbolic links? List any shell commands you used to answer this question.
9. What is the oldest regular file in the `/usr/lib64` directory? Use the last-modified time to determine age. Specify the name of the file without the `/usr/lib64/` prefix. Consider files whose names start with "`.`". List any shell commands you used to answer this question.
10. Where does the `locale` command get its data from? List any shell commands you used to answer this question.
11. In Emacs, what commands have `downcase` in their name? List any Emacs commands you used to answer this question.
12. Briefly, what do the Emacs keystrokes `C-M-r` through `C-M-v` do? Can you list their actions concisely? List any Emacs commands you used to answer this question.
13. In more detail, what does the Emacs keystroke `C-g` do? List any Emacs commands you used to answer this question.
14. What does the Emacs `yank` function do, and how can you easily invoke it using keystrokes? List any Emacs commands you used to answer this question.
15. When looking at the directory `/usr/bin`, what's the difference between the output of the `ls -l` command, and the directory listing of the Emacs `dired` command? List any shell or Emacs commands you used to answer this question.

Homework: Learning to use Emacs

- Keith Waclena, A Tutorial Introduction to GNU Emacs (2009)
- The Emacs editor, version 26.3 (2019)
- An Introduction to Programming in Emacs Lisp, version 26.3 (2019)

For all the exercises, record the steps taken to accomplish the given tasks. Use intelligent ways of answering the questions. For example, if asked to move to the first occurrence of the word "scrumptious", do not merely use cursor keys to move the cursor by hand; instead, use the builtin search capabilities to find "scrumptious" quickly.

To start, download a copy of the web page you're looking at into a file named `assign1.html`. You can do this with Wget or curl. Use `cp` to make three copies of this file. Call the copies `exer1.html`, `exer2.html`, and `exer3.html`.

Exercise 1.1: Moving around in Emacs

1. Use Emacs to edit the file `exer1.html`.
2. Move the cursor to just after the first occurrence of the word "HTML" (all upper-case).
3. Now move the cursor to the start of the first later occurrence of the word "scavenger".
4. Now move the cursor to the start of the first later occurrence of the word "self-referential".
5. Now move the cursor to the start of the first later occurrence of the word "arrow".
6. Now move the cursor to the end of the current line.
7. Now move the cursor to the beginning of the current line.
8. Doing the above tasks with the arrow keys takes many keystrokes, or it involves holding down keys for a long time. Can you think of a way to do it with fewer keystrokes by using some of the commands available in Emacs?
9. Did you move the cursor using the arrow keys? If so, repeat the above steps, without using the arrow keys.
10. When you are done, exit Emacs.

Exercise 1.2: Deleting text in Emacs

1. Use Emacs to edit the file `exer2.html`. The idea is to delete its HTML comments; the resulting page should display the same text as the original.
2. Delete the 18th line, which is an HTML comment. `<!-- HTML comments look like this. -->`
3. Delete the HTML comment containing the text "`DELETE-ME DELETE-ME DELETE-ME`".
4. Delete the HTML comment containing the text "`https://en.wikipedia.org/wiki/HTML_comment#Comments`".
5. There are three more HTML comments; delete them too.

Once again, try to accomplish the tasks using a small number of keystrokes. When you are done, save the file and exit back to the command line. You can check your work by using a browser to view `exer2.html`. Also, check that you haven't deleted something that you want to keep, by using the following command:

```
diff -u exer1.html exer2.html >exer2.diff
```

The output file `exer2.diff` should describe only text that you wanted to remove. Don't remove `exer2.diff`; you'll need it later.

Exercise 1.3: Inserting text in Emacs

1. Use Emacs to edit the file `exer3.html`.
2. Change the first two instances of "Assignment 1" to "Assignment 42".
3. Change the first instance of "UTF-8" to "US-ASCII".
4. Ooops! The file is not ASCII so you need to fix that. Remove every line containing a non-ASCII character. You can find the next non-ASCII character by searching for the regular expression `[^[:ascii:]]`.
5. Insert an empty line after the first line containing "`</ol>`".
6. When you finish, save the text file and exit Emacs. As before, use the `diff` command to check your work.

Exercise 1.4: Other editing tasks in Emacs

In addition to inserting and deleting text, there are other common tasks that you should know, like copy and paste, search and replace, and undo.

1. Execute the command "`cat exer2.html exer2.diff >exer4.html`" to create a file `exer4.html` that contains a copy of `exer2.html` followed by a copy of `exer2.diff`.
2. Use Emacs to edit the file `exer4.html`. The idea is to edit the file so that it looks identical to `exer1.html` on a browser, but the file itself is a little bit different internally.
3. Go to the end of the file. Copy the new lines in the last chunk of diff output, and paste them into the correct location earlier in the file.
4. Repeat the process, until the earlier part of the file is identical to what was in the original.
5. Delete the last part of the file, which contains the diff output.
6. … except we didn't really want to do that, so undo the deletion.
7. Turn the diff output into a comment, by surrounding it with "`<!--`" and "`-->`".
8. Now let's try some search and replaces. Search the text document for the pattern "`<ol>`". How many instances did you find? Use the search and replace function to replace them all with the final-caps equivalent "`<oL>`".
9. Check your work with viewing `exer4.html` with an HTML browser, and by running the shell command "`diff -u exer1.html exer4.html >exer4.diff`". The only differences should be changes from "`<ol>`" to "`<oL>`", and a long HTML comment at the end.

Exercise 1.5: Doing commands in Emacs

Do these tasks all within Emacs. Don't use a shell subcommand if you can avoid it.

1. Create a new directory named "`junk`" that's right under your home directory.
2. In that directory, create a C source file `hello.c` that contains the following text. Take care to get the text exactly right, with no trailing spaces or empty lines, with the initial `#` in the leftmost column of the first line, and with all other lines indented to match exactly as shown:

```
#include <stdio.h>
int
main (void)
{
  int c = getchar ();
  if (c < 0)
    {
      if (ferror (stdin))
        perror ("stdin");
      else
        fprintf (stderr, "EOF on input\n");
      return 1;
    }
  if (putchar (c) < 0 || fclose (stdout) != 0)
    {
      perror ("stdout");
      return 1;
    }
  return 0;
}
```

3. Compile this file, using the Emacs `M-x compile` command.
4. Run the compiled program from Emacs using the `M-!` command, and put the program's output into a new Emacs buffer named `hello-out`.
5. Copy this buffer's contents directly into the log that you're maintaining for this exercise. (You *are* using Emacs to maintain the log, aren't you?)

Exercise 1.6: Running Elisp code

1. Visit Emacs's `*scratch*` buffer.
2. In the buffer, evaluate the two constants `most-negative-fixnum` and `most-positive-fixnum`. Use `C-j` (`eval-print-last-sexp`) to evaluate the constants, and record the results that you get. What is a good explanation for these values? (Hint: convert them to hexadecimal; you can use the Emacs `format` function to do that. Type "`C-h f format RET`" to learn more about this function.)
3. In the buffer, seed the random number generator with your student ID as a string, dashes included. For example, if your student ID is 123-456-789, evaluate `(random "123-456-789")`. As before, use `C-j` to evaluate the expression, and record the result that you get. You can find out more about this function with "`C-h f random RET`".
4. In the buffer, assign two random integers to the global variables `x` and `y`. Start by executing `(setq x (random))`. As usual, use `C-j` and record your results.
5. Compute the product `p` of the two variables by evaluating `(* x y)`. If `p` is negative, compute `(logior p most-negative-fixnum)`; otherwise compute `(logand p most-positive-fixnum)`. Call the result `r`. Record `p` and `r`.
6. Compare `r` to the mathematical product of `x` and `y`. If `r` is the correct mathematical answer, keep trying again with a different pair of random integers until you get an `r` that is not mathematically correct.
7. Are the two integers `x` and `y` truly random in the mathematical sense? If not, what's not random about them?
8. Assuming `(random)` is truly random, what is the probability that the `r` value mentioned above is the mathematically correct product of the two integers? Explain how you calculated this.

Submit

Submit the following two files, in either plain-text or Org format (your choice).

`key1.txt` or `key1.org`
: For each homework exercise, the set of keystrokes needed to do the exercise. Attempt to use as few keystrokes as possible. Do not bother to write down the keystrokes needed to start the editor (e.g., "`e m a c s SP e x e r 1 . h t m l Enter`") or to type in the complicated C program of Exercise 1.5. Write down the label of the key for each keystroke, e.g., "`a`", "`A`" (if you type "`a`" while holding down the shift key), "`Tab`", "`Enter`", "`Esc`". Use "`SP`" for space. Use prefix "`C-`" and "`M-`" for control and meta characters: e.g., "`C-f`" represents Control-F, and "`M-f`" represents Meta-F. Put a space or a newline between each pair of keystroke representations. For example: "`e m a c s < v i Backspace Backspace Backspace > v i`". If you use some key not described above, invent your own ASCII name for the key and explain what key you mean, but don't put spaces or newlines in your key name.

`ans1.txt` or `ans1.org`
: Answers to each lab question.

The files should be ASCII text files with no control characters other than tab and newline (in particular, the files should not contain carriage returns). The shell commands:

```
LC_ALL=C awk '/[^\t\f -~]/' key1.org ans1.org # (if you submit .org files)
LC_ALL=C awk '/[^\t\f -~]/' key1.txt ans1.txt # (if you submit .txt files)
```

should output nothing.

$Id: assign1.html,v 1.49 2020/01/06 17:55:58 eggert Exp $