Assignment 1. Files and shell scripting

[course home > assignments]

This assignment is designed to give you familiarity with two things. The first is Emacs, the classic programmable text editor that is a prototype for modern integrated development environments (IDEs). The second is scripting with POSIX-compatible shells. Software developers should be expert in both IDEs and scripting, even if they don't necessarily use Emacs or shells per se.

Do this assignment on the SEASnet GNU/Linux servers `lnxsrv11`, `lnxsrv12`, `lnxsrv13`, or `lnxsrv15`, with `/usr/local/cs/bin` prepended to your `PATH`. You can do this by executing the shell command "`export PATH=/usr/local/cs/bin:$PATH`" after logging in, or more conveniently by putting that shell command into your `$HOME/.profile` file (but test this file by logging in via a separate session before logging out of your first session!).

As this course has no textbook, a main goal of this assignment is covering how you can discover details about this assignment's topic, details that may not be covered in lecture. Although you can get some of the details by following all the links in this assignment and getting the gist of those web pages, this won't suffice for everything and you'll need to do some learning-by-doing to do the assignment well. The idea is that you can put this experience to good use later in this course (and in real life!) when you need to come up to speed with a large software ecosystem. That being said, don't let yourself get discouraged if a detail cannot be found by reading the online documentation. If you need a hint, ask a TA or an LA. (This assignment is not intended to be done without any hints!)

Laboratory: Linux and Emacs scavenger hunt

- Emacs supplies its own tutorial; try it by running the shell command `emacs` and then type `control-H` followed by `t` ("`C-h t`", in Emacs-ese).
- Keith Waclena, A Tutorial Introduction to GNU Emacs (2009)
- GNU Emacs manual, version 28.2 (2022)

- An Introduction to Programming in Emacs Lisp, version 28.2 (2022)

Instructions: Do the lab part of this assignment (including all shell commands and editing) under Emacs. After issuing the shell command `emacs` to start Emacs, you can run the Emacs `M-x open-dribble-file` command to create a dribble file `lab1.drib` in your home directory that records everything you type. If you do multiple Emacs sessions, name your dribble files `lab2.drib`, `lab3.drib`, etc. (You need to generate dribble files only for this assignment; we won't bother with dribble files in later assignments.)

For the editing lab exercises, 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`.

Lab 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.

Lab 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 69th line, which is an HTML comment. `<!-- HTML comments look like this, but the comment you delete has different text inside. -->`
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 two 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.

Lab 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 27".
3. Change the first instance of "UTF-8" to "US-ASCII".
4. Ooops! The file is not ASCII so you need to fix that. Most of its non-ASCII characters are the Unicode character “’” (RIGHT SINGLE QUOTATION MARK U+2019); fix these by replacing each one with an ASCII “'” (U+0027 APOSTROPHE); for example, you can use `M-x replace-string` to do this systematically.
5. Find the first remaining character in the file that is not ASCII. You can find the next non-ASCII character by searching for the regular expression "`[^[:ascii:]]`".

6. What non-ASCII character is it? You can use `C-u C-x =` (`what-cursor-position`) to find out.
7. Remove every line that contains a non-ASCII character other than the U+2019 characters you already replaced.
8. When you finish, save the text file and exit Emacs. As before, use the `diff` command to check your work.

Lab 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 "`-->`". If the diff output itself contains end-comment markers "`-->`", escape them by replacing each such "`-->`" with "`--&gt;`".
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.

Lab 1.5: Exploring the operating system outside Emacs

Use the commands that you learned in class to find answers to the following questions. Don't use a search engine like Google to find previous editions of this assignment and/or its answers, and don't ask your neighbor, don't use GitHub, etc. When you find a new command, run it so you can see exactly how it works.

1. Where are the `sh`, `sleep`, and `type` commands located?
2. What executable programs in `/usr/bin` have names that are exactly three characters long and start with the two letters `se`, and what do they do?
3. When you execute the command named by the symbolic link `/usr/local/cs/bin/emacs`, which file actually is executed?
4. What is the version number of the `/usr/bin/gcc` program? of the plain `gcc` program? Why are they different programs?
5. The `chmod` program changes permissions on a file. What does the symbolic mode `u+sx,o-w` mean, in terms of permissions?
6. Use the `find` command to find all directories modified on or after the day of this class's first lecture, that are located under (or are the same as) the directory `/usr/local/cs`. If there are more than ten such directories, sort the directory names alphabetically and just just the first ten names.
7. Of the files in the same directory as `find`, how many of them are symbolic links?
8. What is the oldest 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. Don't ignore files whose names start with "`.`", but do ignore files under subdirectories of `/usr/lib64/`. Consider files of all types, that is, your answer might be a regular file, or a directory, or something else.
9. In Emacs, what commands have `transpose` in their name?
10. What does the Emacs `yank` function do, and how can you easily invoke it using keystrokes?
11. 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?
12. Use the `ps` command to find your own login shell's process, all that process's ancestors, and all its descendants. Some `ps` options that you might find useful include `-e`, `-f`, `-j`, `-l`, `-t`, `-H`, and `-T`.

Lab 1.6: 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)
{
  for (;;)
    {
      int c = getchar ();
      if (c < 0)
        {
          if (ferror (stdin))
            perror ("stdin");
          else
            fprintf (stderr, "EOF on input\n");
          return 1;
        }
      if (putchar (c) < 0 || (c == '\n' && fclose (stdout) != 0))
        {
          perror ("stdout");
          return 1;
        }
      if (c == '\n')
        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 standard output into a file named `hello-a1` and its standard error into a file `hello-a2`.
5. Same as before, except run the program with standard input being closed, and put the program's standard output and error into `hello-b1` and `hello-b2`, respectively. Here, "closed" does not mean the standard input is an empty file; it means that standard input is not open at all, to any file.
6. Same as before, except run the program with standard input being the file `/etc/passwd`, and put the program's standard output and error into `hello-c1` and `hello-c2`.
7. Same as before, except run the program with standard input being the file `/etc/passwd` and standard output being the file `/dev/full`, and put the program's standard error into `hello-d2`.

Homework: Scripting in the shell

For the homework assume you're in the standard C or POSIX locale. The shell command `locale` should output `LC_CTYPE="C"` or `LC_CTYPE="POSIX"`. If it doesn't, use the following shell command:

```
export LC_ALL='C'
```

and make sure `locale` outputs the right thing afterwards.

Shell scripting

- Steve Parker, Shell Scripting Tutorial (2022)
- The Open Group, Shell Command Language (2018)

Examine the SEASnet file `/usr/share/dict/linux.words`, which contains a list of English words, one per line. Each word consists of one or more ASCII characters.

Sort this file and put the sorted output into a file `sorted.words`.

Then, take a text file containing the HTML in this assignment's web page, and run the following commands with that text file being standard input. For each command `tr`, `sort`, `comm`, read the command's man page and use that to

deduce what the command should do given its operands here. Also, look generally at what each command outputs (in particular, how its output differs from that of the previous command), and why.

```
tr -c 'A-Za-z' '[\n*]'
tr -cs 'A-Za-z' '[\n*]'
tr -cs 'A-Za-z' '[\n*]' | sort
tr -cs 'A-Za-z' '[\n*]' | sort -u
tr -cs 'A-Za-z' '[\n*]' | sort -u | comm - sorted.words
tr -cs 'A-Za-z' '[\n*]' | sort -u | comm -23 - sorted.words
```

Let's take the last command as the crude implementation of an English spelling checker. This implementation mishandles the input file `/usr/share/dict/linux.words`! Write a shell script named `myspell` that fixes this problem. Your script should read from standard input and output misspelled words to standard output, for a suitable definition of "words". The shell command:

```
./myspell </usr/share/dict/linux.words
```

should output nothing, because the dictionary by definition contains only correctly-spelled words.

If you like, your shell script can also read a file named `my.words`. If it does, write another shell script `makedict` that reads a copy of `/usr/share/dict/linux.words` on standard input and outputs a copy of `my.words` on standard output. That is, you should be able to create `my.words` by running the shell command:

```
./makedict </usr/share/dict/linux.words >my.words
```

Submit

Submit the following files within a compressed tarball named `assign1.tgz`.

- `lab1.drib` and any later dribble files that you generate.
- The `hello-??` files of Lab 1.6.
- `myspell`. This should not have a file name extension; for example, it should not be `myspell.sh`.

- `makedict`. This also should lack a file name extension. Do not submit this file if your `myspell` script can operate without an auxiliary dictionary file to read.
- `notes.txt`, a text file containing any other notes or comments that you'd like us to see.

All files other than the `.drib` files should use GNU/Linux style, i.e., UTF-8 encoding with LF-terminated lines.

The shell command:

```
tar -tvf assign1.tgz
```

should output a list of file names that contains `myspell` etc., with sizes and other metainformation about the files.

$Id: assign1.html,v 1.70 2023/04/02 20:46:20 eggert Exp $