Assignment 2. Shell scripting

Laboratory: Spell-checking Hawaiian

Keep a log in the file `lab2.log` 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.

For this laboratory we 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.

We also assume the file `words` contains a sorted list of English words. Create such a file by sorting the contents of the file `/usr/share/dict/words` on the SEASnet GNU/Linux hosts, and putting the result into a file named `words` in your working directory. To do that, you can use the `sort` command.

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

Let's take the last command as the crude implementation of an English spelling checker. Suppose we want to change it to be a spelling checker for Hawaiian, a language whose traditional orthography has only the following letters (or their capitalized equivalents):

```
p k ' m n w l h a e i o u
```

In this lab for convenience we use ASCII apostrophe (') to represent the Hawaiian ‘okina (‘); it has no capitalized equivalent.

Create in the file `hwords` a simple Hawaiian dictionary containing a copy of all the Hawaiian words in the tables in "English to Hawaiian", an introductory list of words. Use Wget to obtain your copy of that web page. Extract these words systematically from the tables in "English to Hawaiian". Assume that each occurrence of "`<tr> <td>`*`Eword`*`</td> <td>`*`Hword`*`</td>`" contains a Hawaiian word in the *`Hword`* position. Treat upper case letters as if they were lower case; treat "`<u>a</u>`" as if it were "`a`", and similarly for other letters; and treat ` (ASCII grave accent) as if it were `'` (ASCII apostrophe, which we use to represent ‘okina). Some entries, for example "`H<u>a</u>lau, kula`", contain spaces or commas; treat them as multiple words (in this case, as "`halau`" and "`kula`"). You may find that some of the entries are improperly formatted and contain English rather than Hawaiian; to fix this problem reject any entries that contain non-Hawaiian letters after the abovementioned substitutions are performed. Sort the resulting list of words, removing any duplicates. Do not attempt to repair any remaining problems by hand; just use the systematic rules mentioned above. Automate the systematic rules into a shell script `buildwords`, which you should copy into your log; it should read the HTML from standard input and write a sorted list of unique words to standard output. For example, we should be able to run this script with a command like this:

```
cat foo.html bar.html | ./buildwords | less
```

If the shell script has bugs and doesn't do all the work, your log should record in detail each bug it has.

Modify the last shell command shown above so that it checks the spelling of Hawaiian rather than English, under the assumption that `hwords` is a Hawaiian dictionary. Input that is upper case should be lower-cased before it is checked against the dictionary, since the dictionary is in all lower case.

Check your work by running your Hawaiian spelling checker on this web page (which you should also fetch with Wget), and on the Hawaiian dictionary `hwords` itself. Count the number of "misspelled" English and Hawaiian words on this web page, using your spelling checkers. Are there any words that are "misspelled" as English, but not as Hawaiian? or "misspelled" as Hawaiian but not as English? If so, give examples.

Homework: Find improperly marked files

Warning: it will be difficult to do this homework without attending the lab session for hints.

You're working in a project that has lots of text files. Some of them are in plain ASCII; others are in UTF-8, which is a superset of ASCII. None of the files are supposed to contain NUL bytes (all bits zero). The UTF-8 files that contain non-ASCII characters are supposed to have a first line containing the string "`-*- coding: utf-8 -*-`" (without the quotation marks).

POSIX systems typically support various locales to let applications operate well in various countries, languages, and character encodings. The shell command `locale -a` outputs all the locales available on your system, and should list among others the `C`, `es_MX.utf8`, and `ja_JP.utf8` locales. You can select a locale by setting the `LC_ALL` environment variable with a shell command like the following:

```
export LC_ALL=en_US.utf8
```

This setting takes effect for all programs later invoked. You can see your current locale settings by running the `locale` command without any arguments. You can see how the locale affects the output of some standard utilities by running the `date` and `ls -l` commands, using the abovementioned locales.

Associated with each locale is a character encoding that specifies how characters in that locale are represented as bytes. In the `C` locale on GNU/Linux systems, each character represents a single byte using the ASCII encoding for bytes whose values are in the range 0–127; bytes outside of that range do not represent any characters, and are considered to be encoding errors. The `en_US.utf8` locale is like the C locale, except that some (but not all) short sequences of bytes in the range 128–255 represent non-ASCII characters; bytes that are not in such a sequence are considered to be encoding errors.

1. Write a shell script `find-ascii-text` that accepts one or more arguments, and outputs a line for each argument that names an ASCII text file (i.e., an ASCII file containing no NUL bytes). If an argument names a directory, your script should recursively look at all files under that directory or its subdirectories.
2. Write a similar shell script `find-utf-8-text` that works like `find-ascii-text`, except that it outputs a line only for UTF-8 text files (i.e., UTF-8 files containing no NUL bytes) that are not ASCII text files.
3. Write a shell script `find-missing-utf-8-header` that accepts one or more arguments, and outputs a line for each argument naming a UTF-8 text file that lacks the "`-*- coding: utf-8 -*-`" string in the first line. If an argument names a directory, the command should search the directory recursively.
4. Write a shell script `find-extra-utf-8-header` that is like `find-missing-utf-8-header` except it outputs a line for each ASCII text file that has the "`-*- coding: utf-8 -*-`" string in the first line.

You need not worry about the cases where your scripts are given no arguments. However, be prepared to handle files whose names contain special characters like spaces, "*", and leading "–". You need not worry about file names containing newlines.

Your script should be runnable as an ordinary user, and should be portable to any system that supports GNU `grep` along with the other standard POSIX shell and utilities; please see its list of utilities for the commands that your script may use. (Hint: see the `find`, `head` and `tr` utilities.) With GNU `grep`, the pattern `.` (period) matches only individual characters in the current locale; it does not match encoding errors. GNU `grep` has special treatment of files with encoding errors, or files containing NUL characters; see its `--binary-files` option. You may also want to look at GNU `grep`'s `-H`, `-m`, `-n`, `-l`, `-L`, and `-v` options.

When testing your script, it is a good idea to do the testing in a subdirectory devoted just to testing. This will reduce the likelihood of messing up your home directory or main development directory if your script goes haywire.

Submit

Submit the following files. *Warning: You should edit your files with Emacs or with other editors that end lines with* `'\n'`. Do not use Notepad or similar tools that may convert line endings to CRLF form.

- The script `buildwords` as described in the lab.
- The file `lab2.log` as described in the lab.
- The four scripts 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:

```
awk '/\r/ || 80 < length' lab2.log find-ascii-text find-utf-8-text find-missing-utf-8-header find-extra-utf-8-header
```

should output nothing.