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 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 # ENGLISHCHECKER
```

Let's take the last command (marked ENGLISHCHECKER) as the crude implementation of an English spelling checker. Suppose we want to change ENGLISHCHECKER 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 (ʻ); 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 "Hawaiian to English", an introductory list of words. Use Wget to obtain your copy of that web page. Extract these words systematically from the tables in "Hawaiian to English". Remove all instances of '`?`', '`<u>`' and '`</u>`'. For each remaining line of the form '`A<td`*X*`>`*W*`</td>`*Z*', where *A* and *Z* are zero or more spaces, *X* contains no '`>`' characters and *W* consists of entirely Hawaiian characters or spaces, assume that *W* contains zero or more nonempty Hawaiian words and extract those words; each word is a maximal sequence of one or more Hawaiian characters. Treat upper case letters as if they were lower case, and treat `` ` `` (ASCII grave accent) as if it were `'` (ASCII apostrophe, which we use to represent 'okina). For example, the entry '`H<u>a</u>`ule lau' contains the two words '`ha'ule`' and '`lau`'. 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.

From the ENGLISHCHECKER command, derive a shell command HAWAIIANCHECKER that checks the spelling of Hawaiian rather than English, under the assumption that `hwords` is a Hawaiian dictionary and that every maximal nonempty sequence of ASCII letters or apostrophe is intended to be a Hawaiian word and needs its spelling checked. 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 very web page (which you should also fetch with Wget), and on the Hawaiian dictionary `hwords` itself. Count the number of distinct misspelled words on this very web page, using both ENGLISHCHECKER and HAWAIIANCHECKER. Count the number of distinct words on this very web page that ENGLISHCHECKER reports as misspelled but HAWAIIANCHECKER does not, and give two examples of these words. Similarly, count the number of distinct words (and give two examples) that HAWAIIANCHECKER reports as misspelled on this very web page but ENGLISHCHECKER does not.

Homework: Find poorly-named 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 files and is intended to be portable to a wide variety of systems, some of which have fairly-restrictive rules for file names. Your project has established the following portability guidelines:

1. A file name component must contain only ASCII letters, '`.`', '`-`', and '`_`'. A *file name component* is a nonempty part of a file name that is a maximal sequence of characters other than '`/`'; for example, the file name '`/usr/lib64/libstdc++.so.6`' has the three components '`usr`', '`lib64`', and '`libstdc++.so.6`', and the last component's two '`+`' characters violate this guideline.
2. A file name component cannot start with '`-`'.
3. Except for '`.`' and '`..`', a file name component cannot start with '`.`'.
4. A file name component must not contain more than 14 characters.
5. No two files in the same directory can have names that differ only in case. For example, if a directory contains a file '`St._Andrews`' it cannot also contain a file name '`st._anDrEWS`'.

Your boss has given you the job of looking for projects that do not follow these guidelines. Write a shell script `poornames` that accepts a project's directory name *D* as an operand and looks for violations of one or more of the guidelines in files under *D*. There are two parts to this assignment: first, you need to write a basic `poornames` script, and second, you need to extend `poornames` to make it recursive.

Basic `poornames`

In its basic form, when given one operand *D*, `poornames` should look for violations in *D*'s directory entries (i.e., in files immediately under *D*). If *D* is not the name of a directory, `poornames` should diagnose the problem on stderr and exit with a failure status. (If *D* is a symbolic link, `poornames` should not follow the symbolic link, and should treat it as an error just like any other non-directory.) When given no operands, `poornames` should act as if *D* is '`.`', the current working directory. When given two or more operands, or a single operand whose first character is '`-`', `poornames` should diagnose the problem on stderr and exit with a failure status. If `poornames` encounters an error when examining a directory (e.g., lack of permissions to read the directory) it should diagnose the problem on stderr, but it need not exit with a failure status.

The `poornames` script should output a line containing each full file name if and only if the file name's last component does not follow the guidelines. The full file name should start with *D*, followed by '`/`' if *D* does not already end in '`/`', followed by the file name component that does not follow the guidelines, and finally followed

by a trailing '/' if the named file is a directory. The `poornames` script should not output duplicate lines. The order of output lines does not matter.

The `poornames` script should work with file names whose components contain special characters like spaces, '`*`', and leading '`-`' (except that *D* itself cannot begin with '`-`'). However, 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 standard POSIX shell and utilities; please see its list of utilities for the commands that your script may use. (Hint: see the `find` and `sort` utilities.)

Recursive `poornames`

Once you've implemented the basic form, extend `poornames` so that it optionally runs recursively. When given an '`-r`' option-argument, recursive `poornames` should look recursively at every directory entry in every directory under *D*, including *D* itself.

Any '`-r`' option-argument must precede any *D* operand, and at most one option-argument (i.e., argument beginning with '`-`') can be given; otherwise `poornames` should report the problem on standard error and exit with a failure status.

Recursive `poornames` should not follow symbolic links when recursively looking for poorly-named files. However, it should check the symbolic links' names, just as it checks the names of all other files.

If a subdirectory name violates the guidelines, `poornames` should report the name, but this should not affect whether names of files under the subdirectory are also reported.

Although basic '`poornames -r`' should report an error and exit with a failure status, recursive '`poornames -r`' should behave like '`poornames -r .`' and recurse through the working directory. The grader should be able to use use the behavior of '`poornames -r`' to test whether you have attempted to do recursive `poornames`, or just basic `poornames`.

Testing `poornames`

Here are some simple tests to try, but you should not limit yourself to just these tests.

The command '`poornames /usr/bin`' should output a line '`/usr/bin/[`' because the `/usr/bin` directory contains a file `[` that contains a character that falls outside the guidelines. The same command should also output the lines '`/usr/bin/head`' and '`/usr/bin/HEAD`' if both files exist, because the directory entries `head` and `HEAD` fall outside the guidelines (either entry alone would be OK, but the presence of both entries mean that both are invalid). The command '`poornames /usr/share/locale`' should output the line '`/usr/share/locale/en@quot/`' if that directory exists, because '`@`' is not one of the allowed characters.

The commands '`poornames /usr/lib`' should and '`poornames -r /usr/lib`' should both output '`/usr/lib/libstdc++.so.6`' if it exists, because '`+`' is not one of the allowed characters; '`poornames -r /usr/lib`' should also output the lines '`/usr/lib/firmware/radeon/hawaii_ce.bin`' and '`/usr/lib/firmware/radeon/HAWAII_ce.bin`' if those two files exist, because the two directory entries in `/usr/lib/firmware/radeon` are the same if case is ignored.

To illustrate the rules for subdirectories, suppose `d/a`, `d/a/b`, `d/a/B`, `d/a/c`, `d/A`, `d/A/b`, `d/A/c` are all directories, `d/a/C` is a file, and no other files exist. Then '`poornames -r d`' should output lines '`d/a/`', '`d/a/b/`', '`d/a/B/`', '`d/A/`', '`d/a/c/`', and '`d/a/C`', not necessarily in that order; it should not output '`d/A/b/`' because `d/A/b` does not violate any guideline.

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.

Once your script passes your initial tests, try it out as follows:

```
poornames    /usr/bin   > bin.1     2> bin.2
poornames -r /usr/bin   > bin-r.1   2> bin-r.2
poornames    /usr/lib   > lib.1     2> lib.2
poornames -r /usr/lib   > lib-r.1   2> lib-r.2
poornames    /usr/share > share.1   2> share.2
poornames -r /usr/share > share-r.1 2> share-r.2
```

on the SEASnet GNU/Linux host `lnxsrv07`. Save the twelve output files generated by the above commands.

Submit

Submit the following files.

- The script `buildwords` as described in the lab.
- The file `lab2.log` as described in the lab.
- The `poornames` script described in the homework.
- The twelve test output files described in the homework.

Warning: You should edit your files with Emacs or with other editors that end lines with plain LF (i.e., newline or `'\n'`). Do not use Notepad or similar tools that may convert line endings to CRLF form.

All files should be ASCII text files, with no carriage returns, and with no more than 80 columns per line (except possibly for the test output files). The shell command:

```
awk '/\r/ || 80 < length' buildwords lab2.log poornames
```

should output nothing.

$Id: assign2.html,v 1.43 2019/10/10 06:22:57 eggert Exp $