CSE 303, Spring 2005, Assignment 2
Due: Friday 15 April, 9:00AM

Last updated: April 5

In this assignment, you will write and debug shell scripts that use `grep` and `sed`.

1. From the course website, download a *buggy attempt* at the extra extra credit from homework one called `buggy`. Fix the script by changing 9 characters (fixing 6 bugs). Create a *patch* file `prob1patch` such that `patch buggy prob1patch` produces a correct solution. It's okay if the number of characters you change is not exactly 9, but don't rewrite the script; fix it.

2. (htmlgrep) You will write a C-Shell script that follows hypertext links to find strings in (simplified) HTML files and text files. The sample solution is 60 lines, but 22 of the 60 lines are blank or have only comments. The hard parts are using `sed` and setting up the right algorithm. The sample algorithm is described after the problem specification.

   Specification:

   - Call your script `htmlgrep`, taking exactly 3 arguments: `htmlgrep` *`maxdepth file expression`*.
   - There are three types of files: text, html, and other. Using regular-expression syntax, assume all files with names of the form `*.txt` are text, all files with names of the form `*.html?` (i.e., `*.htm` or `*.html`) are html, and everything else is other.
   - Every file has a *depth*. The file passed to `htmlgrep` (call if f) has depth 0. If f is an html file and has a link (described below) to a file g (where $g \neq f$), then g has depth 1. In general, a file has depth i if an html file of depth $i-1$ links to it and no file of depth $< i-1$ links to it. Your script will "examine" (as described below) every file with depth $\leq$ *maxdepth*, in order of increasing depth. For files with the same depth, the order doesn't matter.
   - You may assume all files exist and you have permission to read them. You should not modify them. You can use temporary files (with `mktemp`), but the sample solution doesn't and you should remove any you create.
   - For every file you examine, if you have previously examined it, you should do nothing at all. Else you should first print `f:` (and a newline) where `f` is the name of the file. What you do next depends on the file's type:
     - If it is an other file, print "skipping" and a newline and do nothing else.
     - If it is a text file, call `grep` on the file using the expression passed to `htmlgrep`. (Be sure to quote it.)
     - If it is an html file, do two things:
       * Call `grep` using the expression passed to `htmlgrep` on *a version of the file that has no tags*. That is, strip out all the tags (described below) and call `grep` on the result.
       * Find all the files the file links to and examine them with depth $i+1$.
   - For this problem, the format of html files is very simple:
     - They are mix of *text* and *tags*. Tags start with a `<`, end with a `>` and have no `<`, `>`, or newlines in the middle of them. (The "no newlines" is not true in reality but makes the homework much simpler.)
     - Tags are either links or not links. A link is a tag of the rough form `<a href="filename">`. More specifically, a tag is a link if and only if:
       * The character immediately after `<` is `a` or `A`.
       * There are one or more spaces between `a` and `href="`

    * The word `href` can be spelled with *any combination* of capital and lower-case letters.
    * The filename is every character between the two `"` and will not contain any `"` characters.
    * There can be any number of any character (except `>`) between the second `"` and the `>`.

Warning: It is easy to make this problem a lot more difficult than it is (though it is difficult). Follow the suggested algorithm.

Algorithm:

- Use several variables, which we will call:
  - `curdepth`, the depth of the file currently being examined
  - `thisdepth`, a word-array of files of depth `curdepth` (though some of them may actually have a lower depth)
  - `nextdepth`, a word-array of files of depth `curdepth+1` (though some of them may actually have a lower depth)
  - `seenfiles`, a word-array of files that have already been examined.

  You may need other variables in other places; these are the "key ones".
- To start, `curdepth` is 0, `thisdepth` has one file (the file passed to `htmlgrep`) and the other word-arrays are empty (defined but with 0 elements).
- For each depth d, starting at 0 and going up, examine each file in `thisdepth`:
  - If the file is in `seenfiles`, proceed to the next file, otherwise add it to `seenfiles` (so it won't be processed again).
  - Print out the filename and `:` and a newline.
  - Examine the files extension and proceed appropriately. As described below, if it is an HTML file, we all add any links to `nextdepth`.

  After examining each file of depth d, if $d = maxdepth$, stop. Else increment `curdepth`, make `thisdepth` equal to `nextdepth`, and then reset `nextdepth` to `()`.
- Processing HTML files requires some nontrivial `sed` scripts and some trickery.
  - It is not too difficult to remve all tags and call `grep` on the result. Just remember that tags cannot span lines, but there can be multiple tags on one line. Every `<` starts a tag that continues until the first `>`.
  - Getting the linked files into `nextdepth` is most easily done in two steps: First move every link tag onto its own line (i.e., add a newline character before it and another one after it). Second, take that result and for lines containing links, replace the *entire line* with just the filename (the part between `"` and `"`) and *do not print* the other lines.
  - Hints for `sed`: `-n g p \( \) [ ] ^ * \n [:space:] \1 $`

Extra Credit: Suppose an HTML file is as described above except that newlines *are* allowed in tags. Write a script `movenewlines` that replaces every newline in every tag with a space and then adds the same number of newlines immediately after the tag. (That way, every piece of text is on the same line it used to be. By piping `movenewlines` with `htmlgrep` you have a more useful tool because tags can span lines.) Note: What makes this difficult is the line-oriented nature of `sed`. Warning: The sample solution did not do the extra credit.

Assessment: Your solutions should be:

- Correct shell scripts that run on `attu.cs.washington.edu`
- In good style, including indentation and line breaks
- Of reasonable size

Turn-in Instructions: Follow the link on the course website and follow the instructions there.