COSC 3750

Shell Scripts continued

Kim Buckner

University of Wyoming

Jan. 31, 2023

Last time

- Basic shell syntax

Now what?

- There are two other script-like programs I find helpful.
  - *ed*, the line editor and
  - *sed*, the “stream editor”.
- I am not the greatest with these but they are sometimes very helpful.

The line editor

- Sometimes it is needful to modify a text file from a script.
- The editor that can be used for this is *ed*.
- Part of what makes *ed* useful is that some of you may be familiar with parts of it from VI(M).
- But the <u>problem</u> is that *ed* is a line editor.
- Not a common thing anymore so . . .

Invocation

- `ed filename`
- But this first prints out the number of bytes in the file.
- In a script we will use `ed -s file`, the ‘s’ means silent.
- Then we use basic editing commands you could be familiar with.

Commands

- (.)a – appends text *after* the addressed line.
- The address can be 0, which means that the lines will be added before any others in the file.
- (.)i – inserts text *before* the addressed line. Again, 0 is a valid address for this.
- (.,.)d – deletes the addressed lines.

(more . . .)

- (.,.)c – change. The addressed lines are deleted and the text is inserted in their place.
- (.,.)s/RE/REPLACE/ – The first match of RE on each of the range of lines is replaced with REPLACE.
- A “g” after the command makes it global.
- An integer after the command makes it the the N’th match.

Addresses

- A range like (.,.) can be 2,5
- It can be .,9 where the period means “the current line”.
- It can be 1,$ where the $ means the last line.
- It can also be a single line number.
- The default is just the current line.

Other commands

- (.,.)l – list
- (.,.)p – print, similar to list
- (.,.)n – print with line numbers
- (1,$)w FILE – writes lines to FILE. If no FILE then uses this one.
- If no range then the entire file.
- q – quit. Warned about unwritten changes.
- u – undo last modification.

So what?

- How do we use *ed* in a shell script?
- Use the “here-document” redirection.
- This is input to a command, like it was typed from the keyboard.

Here document

```
<< word
  here-document
delimiter
```

(more . . .)

- The *word* and *delimiter* are the same for our purposes.
- Everything between is taken as lines of text that are input to the command
- The only real restriction on the *delimiter* is that it is unique within the text of the here-document.

Example

```
ed -s myfile << END
0a
This is a new first line.
And this is the second.
.
w
q
END
```

sed

- By default it operates on lines, and does not really care where they come from.
- There are several options, which I almost never use, but you should look at the man page.
- Basically, *sed* is used to modify the input as it passes by.

Regular expressions

- The script is the magic part. A regular expression and a command.
- The *man* page for sed says that the POSIX BREs are supported.
- One place these are described is https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html
- The regular expressions are really same ones you might be familiar with from VI (vim).

(more . . .)

- Characters represent themselves. The asterisk is a modifier that means 0 or more.
- As GNU extensions \+ and \? are available. The **plus** is 1 or more, the **question mark** is 0 or 1.
- The backslashes are **required**.

(more . . .)

- \(\) are used for grouping subexpressions.
- \{J\} is exactly J repetitions of the preceding expression.
- \{J,K\} is at least J but not more than K
- \{J,\} is J or more

(more . . .)

- [ ] enclose a character class.
- [^] reverses the sense of the character class
- There are two "anchors" in these regex (not in character classes)
    - ^ is the beginning of a line and
    - $ is the end of a line (not the newline)

(more . . .)

- The period '.' matches any character including a newline.
- To explicitly match a period you have to use \\.
- The \\| is alternation (or) as in REGEXP\\|REGEXP
- \\DIGIT matches the DIGIT'th subexpression.

(more . . .)

- \n matches the newline (might not be useful)
- But that and \\are the only portable character escapes.
- Specifically, do not depend on \t matching anything but **t**.

The “s” command

- This is substitute and is probably the most used command, at least by me.
- s/REGEXP/REPLACEMENT/FLAGS
- If the REGEXP is matched the REPLACEMENT is substituted for the match.
- The FLAGS can change what happens, for instance “g” means the replacement is done to all matches in the pattern space.

Examples

```
sed ’s/\.tzt/\.txt/’
sed ’s/^\(.*\)\.txt$/PROG_\1_base.tex/’
```