Lecture 8: Using RegExps with Sed and Awk

CS2042 - UNIX Tools

October 17, 2008

Lecture Outline

When to Script

What if we wanted to...

- Change a Notepad-style text file to Unix-style?
- Strip directory prefixes from a path?
- Print certain columns from a text file?
- Remove all the comment lines from some code?

How much time/effort/code would it take to do this stuff in C/C++/Java?

- Stuff like this is the reason scripting was created - you can get each task done in a line or two.

Search and Replace with TR

The Translate Command

tr [options] <set1> [set2]

- Translate or delete characters
- Sets are strings of characters
- Only works on STDIN/STDOUT - use redirection to translate files!

- By default, searches for strings matching *set1* and replaces them with *set2*
- **tr -c <set1> [set2]** will complement *set1* before replacing it with *set2*
- **tr -d <set1>** deletes the characters in *set1* without translating anything.

TR Examples

Example:

Try **echo *** - it prints everything in the directory, separated by spaces. Let's separate them by newlines instead:

- **echo *** | **tr ' ' '/n'** - replaces all spaces with newlines

Example:

Let's print a file in all uppercase:

- **tr 'a-z' 'A-Z'** < **test.txt** - prints the contents of text.txt in all caps

Redirection Revisited

Bash processes I/O redirection from left to right, allowing us to do fun things like this:

Example:

Let's delete everything but the numbers from test1.txt, then store them in test2.txt.

- **tr -cd '0-9'** < **test1.txt** > **test2.txt**

Note:

Redirecting from and to a file at the same time (i.e. if *test2.txt* above were also *test1.txt*) will empty the target file. To preserve the original data, either redirect output to a different file, or append the output to the end of the original using the >> operator.

Lecture Outline

About Sed

Stream Editor

sed [options] [script] [file]

- Stream editor for filtering and transforming text
- We'll focus on the form **sed 's/<regexp>/<text>' [file]**
- This form replaces anything that matches <regexp> with <text>

What is the difference between **sed** and **tr**?

- We can match RegExps!
- **sed** also does a lot of other stuff, grep some docs for details!

Sed In Use

Example:

echo "The sky is blue" > test
sed 's/blue/falling/' test

- `The sky is falling`

Or, without using a file:

Example:

echo "The sky is blue" | sed 's/blue/falling/'

- `The sky is falling`

Lecture Outline

Examples

Let's strip the directory prefix from our pathnames (i.e. convert */usr/local/src* to *src*)

Example:

pwd | sed 's/.*\///'

- Translates anything preceding (and including) a frontslash to nothing
- Note the backslash-escaped frontslash!
- Without escaping the slash, our RegExp would be **.***

A Sed Script

Any text file that begins with #! is a script!

Example:

- Create a new text file named *trim.sed*

```
#!/usr/bin/sed -f
s/^ *//
s/ *$//
```

You can now run this script from the shell like any other local program:

```
echo "      this is a test      " | ./trim.sed
```

- `this is a test`

And we have a script that trims leading and trailing whitespace!

Lecture Outline

A New Language

Awk is another scripting language which is specifically designed to operate on database-like text files.

What is a database?

Databases are files used to store data in a way that is easy to search. A simple database is made up of records (rows) and fields (columns). Fields store data related to each record.

A simple database of highballs and their ingredients might look like:

screwdriver	vodka	orange juice
highball	whiskey	ginger ale
bulldog	gin	grapefruit juice

Awk Syntax

The GNU version of Awk is called **gawk**.

The gawk Command

gawk [options] -f program_file [target_file]
gawk [options] program_text [target_file]

- Programs can be specified through a separate file (1) or as part of the command (2)
- If no target file is specified, it'll work on STDIN

An Awk program consists of a sequence of patterns-actions:

- `pattern {action statements}`

The `action statements` will be run on any input record that matches the `pattern`.

Examples

Print the second and fourth columns of the input

- **echo "This is a test" | gawk '{print $2, $4}'**
    - `is test`
- If no pattern is given, each record is acted on

Print the second column of lines containing *blue*:

- **gawk '/blue/ {print $2}'**

Print lines where the 3rd column is *bologna*

- **gawk '$3=="bologna" {print}'**

Print lines between *#START* and *#FINISH*

- **gawk '/#START/,/#FINISH/ {print}'**

Lecture Outline

User-Defined Variables

Awk Variables

Awk variables are created when they are first used - no declaration necessary. They are either floating-point numbers or strings, with the type based on context.

gawk '{n=5; print $n}'

- Will print the 5th column of each record

Built-in Awk Variables

There are a few Awk built-ins which may be useful to us:

- **FS** - the input field separator - space by default
  - Change this to $\backslash t$ if your file is tab-separated
- **RS** - the input record separator - newline by default
- **NR** - number of input records seen so far
- **NF** - number of fields in the current input record

All of the built-ins can be found in the **gawk** manpage.

BEGIN and END

What if we want to take an action only one time, at the beginning or end of execution?

Special Patterns

BEGIN and END are special patterns which aren't tested against the input. The action parts of all BEGIN patterns are executed before any input is read, and the action parts of END patterns are executed after all input has been exhausted.

Example:

gawk '{sum+=$2} END {print sum}'

- Sums column 2 as input is read, prints the total at the end

gawk '{sum+=$3} END {print (sum/NR)}'

- Prints the average of column 3

Interval Expressions in Awk

Note:

In order to use interval expressions in your RegExps (the things within curly braces, like {**3,**}) you must specify either the **–posix** or **–re-interval** option when you run Awk. The asterisk, question mark, and plus sign operators all work fine by default.