
CSE 374

Programming Concepts & Tools

Hal Perkins

Spring 2010

Lecture 5 – Regular Expressions, grep, Other Utilities

Where we are

- Done learning about the shell and it's bizarre "programming language" (but pick up more on hw3)
- Today: Specifying string patterns for many utilities, particularly grep and sed (also needed for hw3)
- Next: sed, and maybe a little awk
- And then: a real programming language – C

Globbing vs Regular Expressions

- “Globbing” refers to shell filename expansion
- “Regular expressions” are a different but overlapping set of rules for specifying patterns to programs like grep. (Sometimes called “pattern matching”)
- More distinctions:
 - Regular expressions as in CS/mathematics
 - “Regular expressions” in grep
 - “Extended regular expressions” in egrep
 - Same as grep -E
 - Other variations in other programs...

Real Regular Expressions

- Some of the crispest, elegant, most useful CS theory out there. What computer scientists know and ill-educated hackers don't (to their detriment).
- A regular expression p may “match” a string s .
- If $p =$
 - a, b, \dots matches the single character (basic reg. exp.)
 - $p_1 p_2, \dots$, if we can write s as $s_1 s_2$, where p_1 matches s_1 , p_2 matches s_2 .
 - $p_1 | p_2, \dots$ if p_1 matches s or p_2 matches s
 - (in egrep, for grep use $\backslash|$)
 - p_1^* , if there is an $i \geq 0$ such that $p_1 \dots p_1$ (i times) matches s .
 - (for $i = 0$, matches the zero-character string ε)

Conveniences

- Most regular expressions allow various abbreviations for convenience, but these do not make the language any more powerful
 - p^+ is pp^*
 - $p?$ is $(\epsilon \mid p)$
 - $[zd-h]$ is $z \mid d \mid e \mid f \mid g \mid h$
 - $[\wedge a-z]$ and $.$ are more complex, but just technical conveniences
 - $p\{n\}$ is $p \dots p$ (p repeated n times)
 - $p\{n,\}$ is $p \dots pp^*$ (p repeated n or more times)
 - $p\{n,m\}$ is p repeated n through m times

grep – beginning and end of lines

- By default, grep matches each line against `.*p.*`
- You can anchor the pattern with `^` (beginning) and/or `$` (end) or both (match whole line exactly)
- These are still “real” regular expressions

Gotchas

- Modern (i.e., gnu) versions of grep and egrep implement the same regular expression engine, but the syntax is different for historical reasons
 - For instance, \{ for grep vs { for egrep
 - See grep manual sec. 5.6
- Must quote patterns so the shell does not muck with them – and use single quotes if they contain \$ (why?)
- Must escape special characters with \ if you need them literally: \. and . are very different
 - But inside [] many more characters are treated literally, needing less quoting (\ becomes a literal!)

Previous matches – back references

- Up to 9 times in a pattern, you can group with (p) and refer to the matched text later!
 - (Need backslashes in sed.)
- You can refer to the text (most recently) matched by the n^{th} group with $\backslash n$.
- Simple example: double-words `^\([a-zA-Z]*\) \1$`
- You cannot do this with actual regular expressions; the program must keep the previous strings.
 - Especially useful with sed because of substitutions.

Other utilities

- Some very useful programs you can learn on your own:
 - find (search for files, e.g., `find /usr -name words`)
 - diff (compare two files' contents; output is easy for humans and programs to read (see patch))
- Also:
 - For many programs the `-r` flag makes them recursive (apply to all files, subdirectories, subsubdirectories, ...).
 - So “delete everything on the computer” is `cd /; rm -rf *` (be careful!)