

CSE 374 Lecture 6

Regular expressions and grep



Test #1: Linux

- Published today, 11:30am
 - On Canvas
- Due Wed. 1/19 11:00pm
- NO LATE SUBMISSIONS
- Work on your own, but
- Use all available resources
- Hints
 - Catch up on lectures/HW first
 - You can submit up to 3 times
 - First try - take notes of questions that challenge you
 - Look up answers
 - Test code in your shell before answering

Globbing and Regex

Globbing: the shell filename expansion; matches some patterns

Regular expressions (regex): a set of rules for matching patterns in text

(We see regular expressions in math, as formal grammars in cs, and other variations as well. Different applications (egrep) may have slightly different rules.)

Regex theory

1. A set of rules for matching a pattern (P) to a string (S)
2. All strings are made of a combination of the null (empty) set, the empty string ε , and a single character.
3. Regular expressions match a string if
 - 3.1. P is a literal character (a, b, ...) that matches the string S
 - 3.2. P_1P_2 matches S if $S = S_1S_2$ such that P_1 matches S_1 and P_2 matches S_2
 - 3.3. $P_1|P_2$ matches S if P_1 matches S OR P_2 matches S
 - 3.4. P^* matches S if there is an i such that $P...P$ (i times) matches S.
Includes $i=0$ which matches ε .

Regex rules

Regular expressions have

Characters: the literal characters [a b 9] (S is an exact duplicate of P)

Anchors: sets the position in the line where P may be found (^ or \$)

Modifiers: modify the range of text P may match (* or [set_of_chars])

Note: Regex details & implementation may vary between application, but general rules apply.

Regex special characters (*some are \escaped*)

\ : escape following character

'.' : matches any single character at least once

$p_1|p_2$: matches p_1 OR p_2

'*' : matches zero or more of the previous p

'?' : matches zero or one of the previous p ($\epsilon|p$)

'+' : matches one or more of previous p (pp^*)

() : group patterns for order of operations

{ } :

[] : contain literals to be matched (single or range)

^ : Anchors to beginning of line

\$: anchors to end of line

<> : word boundaries

Classes of characters

`.` == any character

`[a-z]` == a, b, c, d ... z

`[A-Z]` == A, B, C, D ... Z

`[0-9]` == 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

`abc` == literally abc

`c.t` → cat, cut, cota

`[Hh]ello!` → Hello!, hello!

`[BLERG]` → B, L, E, R or G

`[0-5][5-9]` → 15, 16, 17, 18, 19
25, 26, 27, 28, 29
35, 36, 37, 38, 39
45, 46, 47, 48, 49
55, 56, 57, 58, 59

Repetition

$*$ == zero or more, $a^* \rightarrow \{, a, aa, aaa, aaaa, \dots\}$

$+$ == one or more, $a^+ \rightarrow \{a, aa, aaa, \dots\}$

$?$ == zero or one of the preceding, $a? \rightarrow \{, a\}$

$\{n\}$ == exactly n repetitions of the preceding, $a\{3\} \rightarrow aaa$

$a|b$ == a or b , $this|that|when|how \rightarrow this, that, when, how$

All but $|$ are **POSTFIX OPERATORS** (they come after the pattern)

Invisible characters

`^` == the start of a line

`$` == the end of a line

`\t` == a tab

`?` == *zero or one* of the preceding

Extras

`[^abc]` : matches everything NOT abc

`*` : is greedy; matches as much as possible

grep

(run `man grep` now!)

Grep - a program to do matching using regular expressions

Grep -e / egrep - uses extended regex

Grep Regex

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

Backreference & repeated matches

Up to 9 times in a pattern, you can group with (p) and refer to the matched text later!

You can refer to the text (most recently) matched by the nth group with \n.

Simple example: double-words `^\([a-zA-Z]*\)\\1$`

You cannot do this with actual regular expressions; the program must keep the previous strings.

`\(p\)\\{n\}` will match the p n times. `\{n,m\}` matches at least n, but not more than m times.

Bash Regex Gotcha's

- Modern (i.e., gnu) versions of grep and egrep use the same regular expression engine for matching, but the input syntax is different for historical reasons
 - For instance, \{ for grep vs { for egrep – See grep manual sec. 3.6
- 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!)