CSE 303
Concepts and Tools for Software Development

Magdalena Balazinska
Winter 2010
Lecture 5 – Regular Expressions and Grep

Outline

- All about regular expressions
- Specifying string patterns for many utilities, particularly grep (today) and sed (next lecture)

Context

- “Globbing” refers to filename expansion characters

- “Regular expressions” are a different but overlapping set of rules for specifying patterns to programs like grep. (Sometimes called “pattern matching”.)

What is a Regular Expression?

```
"[a-zA-Z_\-]+@(([a-zA-Z_\-])+\.)+[a-zA-Z]{2,4}"
```

Regular expression ("regex"): a description of a pattern of text

- Can test whether a string matches the expression's pattern
- Can use a regex to search/replace characters in a string
- Regular expressions are extremely powerful but tough to read
  - (the above regular expression matches basic email addresses)

Regular expressions occur in many places:

- Shell commands (`grep`)
- Many text editors allow regexes in search/replace
- Java `Scanner`

Egrep and Regexes

command	description
`egrep`	extended `grep`; uses regexes in its search patterns; equivalent to `grep -E`

```
egrep "[0-9]{3}-[0-9]{3}-[0-9]{4}" faculty.html
```

`-i` option before regex signifies a case-insensitive match

- `egrep –i “cost”` matches “Costas”, “accosted”, “COSTCO”, ...

Basic Regexes

`"abc"`

- The simplest regexes just match a particular substring
- The above regex matches any line containing "abc"

  YES : `"abc",  "abcdef",  "defabc",  ".=.abc.=.",  ...`

  NO : `"fedcba",  "ab c",  "AbC",  "Bash"`, ...

Wildcards and Anchors

`.` (a dot) matches any character except `\n`

`".oo.y"` matches `"Doocy"`, `"goofy"`, `"LooPy"`, ...

use `\.` to literally match a dot `.` character

`^` matches the beginning of a line; `$` the end

`"^fi$"` matches lines that consist entirely of `fi`

`\<` demands that pattern is the beginning of a *word*;
`\>` demands that pattern is the end of a word

`"\<for\>"` matches lines that contain the word `"for`"

Careful: can easily match beginning of one word and end of another

Special characters

| means OR

- `"abc|def|g"` matches lines with "abc", "def", or "g"
- precedence of `^(Subject|Date):` vs. `^Subject|Date:`
- There's no AND symbol. Why not?

( ) are for grouping

- `"(Homer|Marge) Simpson"` matches lines containing `"Homer Simpson"` or `"Marge Simpson"`

\ escape special characters

- many characters must be escaped to match them: / \ $. [ ] ( ) ^ * + ?
- `"\.\\n"` matches lines containing `".\n"`

Quantifiers: * + ?

`*` means 0 or more occurrences

"`abc*`" matches "ab", "abc", "abcc", "abccc", ...

"`a(bc)*`" matches "a", "abc", "abcbc", "abcbcbc", ...

"`a.*a`" matches "aa", "aba", "a8qa", "a!?_a", ...

`+` means 1 or more occurrences

"`a(bc)+`" matches "abc", "abcbc", "abcbcbc", ...

"`Goo+gle`" matches "Google", "Gooogle", "Goooogle", ...

`?` means 0 or 1 occurrences

"`Martina?`" matches lines with "Martin" or "Martina"

"`Dan(iel)?`" matches lines with "Dan" or "Daniel"

More quantifiers

`{`***`min`***`,`***`max`***`}` means between ***min*** and ***max*** occurrences

- `"a(bc){2,4}"` matches "abcbc", "abcbcbc", or "abcbcbcbc"

min or ***max*** may be omitted to specify any number

- `"{2,}"` means 2 or more
- `"{,6}"` means up to 6
- `"{3}"` means exactly 3

Character Sets

[ ] group characters into a character set;
will match any single character from the set

`"[bcd]art"` matches strings containing "bart", "cart", and "dart"

equivalent to `"(b|c|d)art"` but shorter

Inside [ ], most modifier keys act as normal characters

`"what[.!*?]*"` matches "what", "what.", "what!", "what?**!", ...

Character Ranges

Inside a character set, specify a range of characters with -

- "`[a-z]`" matches any lowercase letter
- "`[a-zA-Z0-9]`" matches any lower- or uppercase letter or digit

An initial ^ inside a character set negates it

- "`[^abcd]`" matches any character other than a, b, c, or d

Inside a character set, `-` must be escaped to be matched

- "`[+\-]?[0-9]+`" matches optional + or -, followed by ≥ one digit

Previous Matches

- The expression \n where n is a number, matches the contents of the n'th set of parentheses in the expression
  - Can do that up to 9 times in a pattern
- Simple example: double-words ^\([a-zA-Z]*\)\1$
- You cannot do this with regular expressions
  - The program must keep the previous strings
- Especially useful with sed because of substitutions

Readings

- Linux Pocket Guide
  - Section about egrep (p. 73-74)