Regular Expressions

CIS*2750

Advanced Programming Techniques

Material for this lectures has been taken from the excellent book, *Mastering Regular Expressions*, 2nd Edition by Jeffrey Friedl.

Regular Expressions

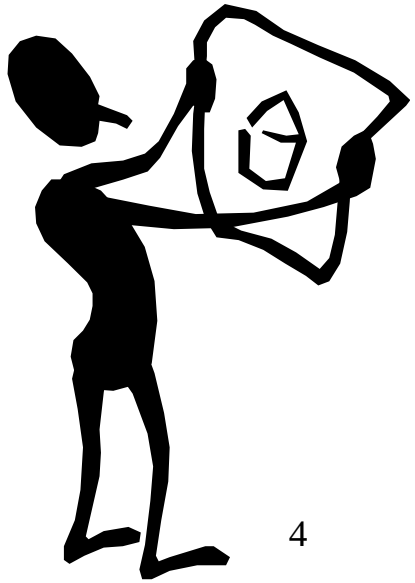
- Regular expressions are a powerful tool for text processing.
- They allow for the description and **parsing** of text.
- With additional support, tools employing regular expressions can **modify** text and data.

Example Problems

- Check many files to confirm that in each file the word **SetSize** appears exactly as often as **ResetSize**. Pay attention to lower and upper case.
  - similar to the problem of checking that your code has as many closing braces (}) as it has opening braces ({)
- Check any number of files and report any occurrence of double words, *e.g.* **the the**
  - should be case insensitive, *i.e.* **the The**
  - if you were checking HTML code then you would have to disregard HTML tags, *e.g.* **the <B>the**

Intriguing reading

- A particularly interesting view of regular expressions can be found in the article, *Marshall McLuhan vs. Marshalling Regular Expressions*.
- Link available on schedule page



Unix Tools: *egrep*

- Given a regular expression and files to search, *egrep* attempts to match the regex to each line of each file.

  egrep '^(From|Search): ' email-file

- *egrep* breaks the input file into separate text lines.
- There is no understanding of high-level concepts, such as **words**.

Examples

- Parsing mail headers in bulk e-mail file

```
egrep '^(From|Subject): ' testmail
From: Deb Stacey dastacey@lashley.cis.uoguelph.ca
Subject: test message
From: Deb Stacey dastacey@lashley.cis.uoguelph.ca
Subject: html doc
From: Deb Stacey dastacey@lashley.cis.uoguelph.ca
Subject: html file
From: "Colleen O'Brien" colleen@snowhite.cis.uoguelph.ca
Subject: Me
From: Deborah Stacey dastacey@snowhite.cis.uoguelph.ca
Subject: PS file (fwd)
From: Deb Stacey dastacey@lashley.cis.uoguelph.ca
Subject: PS file
```

Examples

- The following instance of *egrep* will match the three letters **cat** wherever they are.

egrep 'cat' file

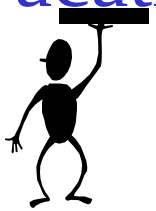
- If a file contains the lines

  While on vacation we saw

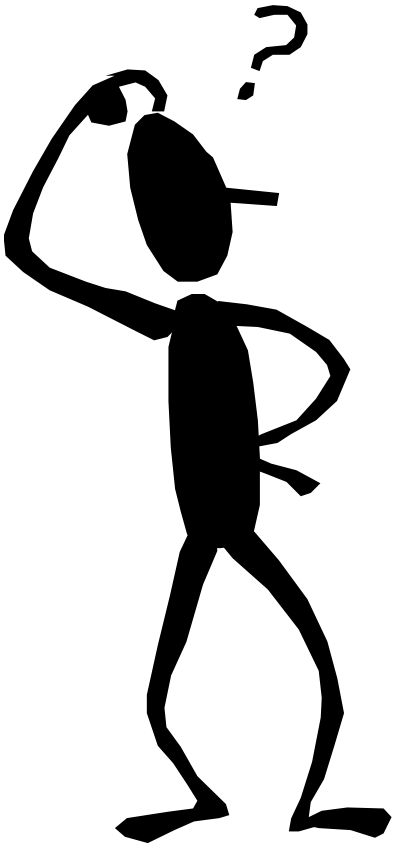
  a fat dog on the beach.

then egrep 'cat' file will produce the result

While on vacation we saw



What do the following mean in *egrep*?



^cat$

^$

^

The Language of Regular Expressions

- **Character**
  - Most text-processing systems treat ASCII 8-bit bytes as characters (including our assignments)
  - Recently, *Unicode* was introduced:
    - Needed for non-Roman foreign languages
    - One Unicode “character” can be 1-4 bytes!
    - Depends on encoding method, UTF-8, -16, -32
    - Requires extensions to regex tools, will come back to Unicode at the end…

Metacharacter

- a *metacharacter* has a special meaning in some circumstances within regular expressions
- a *metacharacter* loses its special meaning when it has been *escaped*
- the *escape* character is usually the backslash \
  - * means the character *
  - * means the character \ followed by *metacharacter* *

Subexpression

- a *subexpression* is a part of a larger expression and is usually within parentheses or are the alternatives of a | clause
- they are particularly important when parsing the meaning of *quantifiers*

Metacharacters

^ *(carat)*

– matches the start of the line

$ *(dollar)*

– matches the end of the line

. *(dot)*

– matches any one character

Metacharacters

[ ] *(character class)*

- matches any character listed in the brackets
- ranges like [a-z] [0-9] are allowed

[^] *(negated character class)*

- matches any character *not* listed after ^

| *(or, bar)*

- matches either expression it separates

( ) *(parentheses)*

- used to designate scope

Negation

- A file contains the following:

While on vacation we saw
a fat dog on the beach.
There was also quite a fat duck
which went 'Quack, quack'!
The beach was in Iraq and so the
dog must have been Iraqian.

- The regex is

  egrep '[Qq][^u]' file

- The result is

The beach was in Iraq and so the
dog must have been Iraqian.

Tricky Regex

- Let's create a file with the following contents:

  Quack

  Iraq

  Qantas

  Quack

- What will the following produce?

egrep '[Qq][^u]' regfile

- It will produce

  Qantas

- What will the following produce?

egrep '[Qq]([^u]|$)' regfile

- It will produce

  Iraq

  Qantas

WHY?

Match Any Character

- Match the date: 07/04/76 or 07-04-76 or 07.04.76
- Regex - version 1

  07[-./]04[-./]76
- Regex - version 2

  07.04.76
- Version 1 [-./]
  - Dot is *not* a metacharacter here:
    inside [ ] class it is just the character dot!
  - Dash is *not* the range metacharacter if *first* in a class!
- Version 2 .
  - The dots are metacharacters that match any single character.

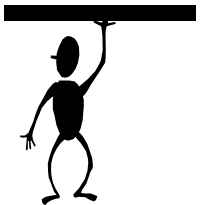
Match Any Character

- If we use egrep '07.04.76' <file> on the file

This strange happening was on the following
date: 07/04/76.
This date could also be written as 07-04-76
or 07.04.76 but who's counting!
My id number is 207 04076 234.

it gives

date: 07/04/76.
This date could also be written as 07-04-76
or 07.04.76 but who's counting!
My id number is 207 04076 234.



WHY?

Alternation

- Search a file for the word **gray** or **grey**.
- Possible (equivalent) regex's are

gr[ea]y

grey|gray

gr(a|e)y

- The difference between **class** (**[ ]**) and **alternation** is that **class** concerns only single characters while alternatives can be regex's on their own.
- What would the following do?

egrep '(From|Subject|Date): ' email-file

Optional Items

- The metacharacter ? appears *after* an optional item.
- colou?r matches color and colour.
- **Problem**: Match any variation on the date July 1, such as Jul 1st, July first, Jul first, etc.

July? (first|1(st)?)

Quantifiers

These all come *after* the expression they quantify!

Meta-Char	*Minimum Required*	*Maximum Allowed*
?	none	one
*****	none	no limit
+	one	no limit

+ means “match one or more of the preceding item”

* means “match zero or more of the preceding item”

Problem

- Match all legal variations of the <Hx> HTML tags

<H[1-6]>

- This works, but allows no optional spaces inside the angle brackets which HTML does.

< *H[1-6] *>

- Matches

<H1>

<H2 >

< H4 >

Problem

- What does

< *HR +SIZE *= *14 *>

match?

- How about

< *HR +SIZE *= *[0-9]+ *>

and

< *HR(+SIZE *= *[0-9]+)? *>

Case Sensitivity in egrep

- To make an expression case insensitive

  egrep -i '< *HR(+SIZE *= *[0-9]+)? *>' file

- and in *Perl* it is **/i**

Problem

- Match all dollar amounts in text including optional cents.

\$[0-9]+(\.[0-9][0-9])?

- **But** will this match $.49 ?
- **NO --** but will the following work?

\$[0-9]*(\.[0-9][0-9])?

- Now allows $ by itself – how about alternation?

\$[0-9]+(\.[0-9][0-9])?|\$\.[0-9][0-9]

Try it yourself

- To discover **why** use the following file (name it *money*):

$10.49
$0.49
$.49
$10
$.
$
$abc

- and test the following:

```
egrep '\$[0-9]+(\.[0-9][0-9])?' money
egrep '\$[0-9]*(\.[0-9][0-9])?' money
egrep '\$[0-9]+(\.[0-9][0-9])?|\$\.[0-9][0-9]' money
```

What is Unicode? (simplified view)

- Defines unique 21-bit number (U+*xxxx*) for every language's characters, called "code points"
  - Represented directly using UTF-32 (with some waste)
  - With shorter representations, UTF-8 and -16, need 1-4 byte sequences for code points > 8 or 16 bits
  - UTF-8 is backward compatible with ASCII (if you don't use its Latin-1 characters)
- Supported by 16-bit **char** in Java/C#
  - C's *wide* **wchar_t**, better support in C11
- In regex, **x** matches one Unicode char (".")

Can we just ignore Unicode?

- "Like we're doing in the assignments"
  - Maybe if you stay in N. America…
- Internationalization of software
  - Make product attractive for non-English users
  - **Localization** → customizing an "internationalized" product to a locale
    - Menus, commands, popups, prompts, etc.
- Unicode important for non-English text processing
  - C is relatively painful for this purpose