Regular Expressions
A Ubiquitous Tool for Manipulating Text

The Executive Summary

Regular expressions are
• a way of representing patterns in text

Why are they useful?
• finding text that matches a pattern
• replacing it with something else
• — saving you time

Where do you find them?
• major word processors (e.g. MS Word)
  obscure keyword: “wildcards”
• most text editors (eg BBEdit, TextWrangler, TextPad, vi, emacs)
  obscure keyword: “grep”
• various command line tools
  UNIX grep, find
  Windows XP’s findstr
• etc, etc, etc...
To import this data into a FileMaker table, we need

• separate fields for given & family names
• a separate field for the *'s flagging privacy

So we want to

• replace

  • • • :*Ferraro, David Joseph: • • •
  • • • :Shillington, Tara Dawn: • • •

• by

  • • • :*David Joseph:Ferraro: • • •
  • • • :*Tara Dawn:Shillington: • • •

(1) These student ID numbers were randomly generated.

Example Problem 2 — Changing a Large Number of Files

<table>
<thead>
<tr>
<th>Change</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC 741 20061112 1500.m4a</td>
<td>Baroque 01.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1501.m4a</td>
<td>Baroque 02.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1502.1.m4a</td>
<td>Baroque 03.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1502.m4a</td>
<td>Baroque 04.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1503.m4a</td>
<td>Baroque 05.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1504.1.m4a</td>
<td>Baroque 06.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1504.m4a</td>
<td>Baroque 07.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1505.m4a</td>
<td>Baroque 08.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1506.m4a</td>
<td>Baroque 09.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1507.m4a</td>
<td>Baroque 10.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1507.1.m4a</td>
<td>Baroque 11.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1508.m4a</td>
<td>Baroque 12.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1509.m4a</td>
<td>Baroque 13.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1509.1.m4a</td>
<td>Baroque 14.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1510.m4a</td>
<td>Baroque 15.m4a</td>
</tr>
<tr>
<td>RC 741 20061112 1511.m4a</td>
<td>Baroque 16.m4a</td>
</tr>
</tbody>
</table>

These files having been generated by Audio Hijack Pro as it records music from a digital cable channel (to play in my car via an iPod)

That is, for convenience we want to change

• a sequence of date-time stamped file names generated by an audio recording program

  RC 741 20061112 1500.m4a

• to a more usable sequence having the form
Example Problem 3 — Change Date Format

We want to change

• dates represented in the European “Day Month Year” style
  25 Sep 1986
• to a format more suitable for importation into an SQL database
  09–25–1986

Our First Regular Expression

Source text:
He is a rat.
She is in a rut.
The food is rotten.
I like root beer.

What’s found (matched strings are underlined and in red):
He is a rat.
She is in a rut.
The food is rotten.
I like root beer.

Discussion
• most characters just match themselves (eg “r” and “t” above)
  in particular, all letters and digits
• a few “special” or “meta” characters are interpreted differently
  “.” (period) matches any single character except the end-of-line character
  (unless the “match newline and carriage return” flag, aka the “dot all” flag, is set)
  “[“ and “]” also have a special meaning, explained on the next slide
to make them non-special, precede them with a backslash — that is, use
  \\. \\[ \\]
Character Classes

Source text:  
My white shirt has turned grey.
She was gray with fright.
“Greyling” is a Polaris server.
My dog is a greyhound.

What’s found:  
My white shirt has turned grey.
She was gray with fright.
“Greyling” is a Polaris server.
My dog is a greyhound.

Discussion

• [ae] matches either “a” or “e”
• Grey is not matched because g and G are different characters
  (although there’s usually a flag you can set for “case-insensitive matching”)
• the pattern matcher doesn’t notice word boundaries
  (although there’s a special “anchor” pattern (\w) that matches word boundaries)

Other examples (note the use of “–” to denote a range of characters)

[0–9] matches any of the ten decimal digits (equivalent to \d — this gives the d special meaning)
[\alpha–\zeta] matches any of the twenty six lowercase alphabetic characters

More Special Characters — ^ * + — and Replacing the Matched

NB: we’ll use ⊔ to represent a blank, so it can be seen and counted

The matched source text (matches shown in red and underlined):
45 There was silence for 17 minutes, and then...
5  Help!
114 Let us go then, you and I, while the evening is laid out against the sky, like a patient, etherized upon a table.
254 Some say the world will end in fire. Some say in ice. From what I’ve tasted of desire, I’d hold with those that favour fire. But if it had to perish twice, I think I know enough of life, to say that for destruction ice is also great, and would suffice.

The result, after the matched patterns have been replaced by the replacement pattern (nothing, in this case):
There was silence for 17 minutes, and then...
Help!
Let us go then, you and I, while the evening is laid out against the sky, like a patient, etherized upon a table.
Some say the world will end in fire. Some say in ice. From what I’ve tasted of desire, I’d hold with those that favour fire. But if it had to perish twice, I think I know enough of life, to say that for destruction ice is also great, and would suffice.

Discussion

“^” matches the beginning of a line
“*” means “0 or more repetitions of the preceding character”

More concise but equivalent patterns

^[0–9]+ + means “one or more repetitions of the preceding character”
\d+ matches any digit, so “\d+” matches any string of one or more digits followed by a
Excluding Characters

Source text:

What’s matched:

Discussion

• ^[ab0-3] matches any character except a, b, 0, 1, 2, or 3
  • ^ has this meaning only at the beginning of a character class — ie, when it’s immediately after a [

Question: could we instead have used the following pattern to match URLs in HTML?

".+"

*hint: + and * are “greedy” — they match the longest string they can*

Matching the end of a line: $ 

What does this pattern/replacement pair do?

Remember that we’re using ⊔ to represent a blank

<table>
<thead>
<tr>
<th>pattern:</th>
<th>$+⊔</th>
</tr>
</thead>
<tbody>
<tr>
<td>replacement:</td>
<td>$+⊔</td>
</tr>
</tbody>
</table>
Sub / Replacement Patterns

Source text:

bmzister 21DalB
bbunny 22BunB
hvbingen 31BinH

What's found:

<TR><TD>bmzister</TD><TD>21DalB</TD></TR>
<TR><TD>bbunny</TD><TD>22BunB</TD></TR>
<TR><TD>hvbingen</TD><TD>31BinH</TD></TR>

Discussion

• typically up to nine subpatterns (\1 through \9)
• “&” represents the entire matched pattern
• why is it unnecessary to surround the pattern with “^” and “$”?

Another Invisible Character — tab — and \w

Source text:

bmzister 21DalB
bbunny 22BunB
hvbingen 31BinH

What's found:

bmzister 21DalB
bbunny 22BunB
hvbingen 31BinH

Discussion

\w [a–zA–Z0–9_]
\t represents the tab character

\w is more general than what we see — is that ok?

Other invisible (aka non-printing) characters

\s white space (blank, tab, newline, carriage return, form feed—ie the character class [\s\t\n\r\f])
\r Macintosh end-of-line (the “carriage return” character)—often represented by ¶
\n Unix end-of-line (the “newline” character)—often represented by ¶
\n\n Windows end-of-line — often represented by ¶
**Alternation: |**

### Source text:

Jack and Jill  
Went up the hill  
To fetch a pail of water.  
Jack fell down  
And broke his crown  
And Jill came tumbling after.

### What's found:

"Jack" and "Jill"  
Went up the hill  
To fetch a pail of water.  
"Jack" fell down  
And broke his crown  
And "Jill" came tumbling after.

### Discussion

\& represents the entire string matched  
so the result encloses the names Jack and Jill in double quotes

### What strings does the following pattern match?

```regex
((Jack|Jill)⊔ran⊔very(ˌ,ˌvery)*⊔fast⊔up⊔the hill!⊔)+
```

---

**A Solution For Problem 1**

### The pattern

```regex
:([^:,.]*),(\[^:,.]*),\s\([^:,.]*\)
```

- p? matches 0 or 1 instances of p
- p* matches 0 or more instances of p
- \* matches " (ie without its special meaning)
- \^[.:] matches any character except : and ,
- \s matches a space
- (p) matches p and remembers what it matches

### The replacement

```regex
\1 is replaced by the 1st ( ) pattern matched \(^\|^\)
\2 is replaced by the 2nd ( ) pattern matched \(^^[.:]*\)
\3 is replaced by the 3rd ( ) pattern matched \(^^[.:]*\)
```

---

**NB:**  
match data is now sorted into increasing order based on the value of this field.  
there's actually another line of "Unmatched Replacement" for each of these — double-click on a row to see it.  

(RegExplorer, which is available for both Macs & Windows.)
Applying This Solution in TextWrangler

Search > Find...

Find & Replace

Search For: O 5

Patterns: 0

Use Grep

Find

Replace

Find All

Replace All

Replace

Repair

Case Sensitive

Match Entire Words

[Image of TextWrangler interface]

Unfortunately, Word’s pattern syntax is deficient & non-standard ...

e.g. no (0 or 1 instance) and no | (alternation)

Pattern 1 (no privacy flag * present)

```plaintext
:([1;]*);,([1;]*);:
```

- `p` means 1 or more instances of `p`
- `[xyz]` means any of the characters `x`, `y`, `z`
- `[!xyz]` means any character except `x`, `y`, `z`
- `(p)` matches `p` and remembers what it matches

Replacement 1

```plaintext
:\1:\1:
```

- `\1` holds the first sub-pattern (p) matched
- `\2` holds the second sub-pattern matched

Solving Problem 1 in MS Word — Step 1

Unfortunately, Word’s pattern syntax is deficient & non-standard ...

e.g. no (0 or 1 instance) and no | (alternation)

Pattern 1 (no privacy flag * present)

```plaintext
:([1;]*);,([1;]*);:
```

- `p` means 1 or more instances of `p`
- `[xyz]` means any of the characters `x`, `y`, `z`
- `[!xyz]` means any character except `x`, `y`, `z`
- `(p)` matches `p` and remembers what it matches

Replacement 1

```plaintext
:\1:\1:
```

- `\1` holds the first sub-pattern (p) matched
- `\2` holds the second sub-pattern matched
Regular Expressions

Pattern 1 (privacy flag * present)
\*\ matches just * (ie, * has no special meaning)
p@\ means 1 or more instances of p
[xyz]\ means any of the characters x, y, z
[!xyz]\ means any character except x, y, z
(p)\ matches p and remembers what it matches

Replacement 1
\1\ holds the first sub-pattern (p) matched
\2\ holds the second sub-pattern matched

Example:
0035091:ygavet::Yann:Gavet::math:NN:ND
95011647:awowkodaw::Andrij:Wowkodaw:sy de:eng:4B:H

A Solution For Problem 2 — Step 1: Prepend a Sequence Number

Change
RC 741 20061112 1500.m4a
RC 741 20061112 1501.m4a
RC 741 20061112 1502.1.m4a

Step 1
01RC 741 20061112 1500.m4a
02RC 741 20061112 1501.m4a
03RC 741 20061112 1502.m4a

Step 2
To Baroque 01.m4a
To Baroque 02.m4a
To Baroque 03.m4a

("A Better Finder Rename," which is available for both Macs & Windows.)
A Solution For Problem 2 — Step 2: Shorten Prefix & Move the

Change | To
---|---
RC 741 20061112 1500.m4a | Baroque 01.m4a
RC 741 20061112 1501.m4a | Baroque 02.m4a
RC 741 20061112 1502.1.m4a | Baroque 03.m4a

Step 1

Step 2

The pattern

\^\(\d\d\)RC 741 20061112 \d\{4,4\}\(\d\d\)\d\$  

\^ matches the beginning of the string
\(\d\d\) matches exactly two digits (0-9)
and remembers them (as \1)
\d\{4,4\} matches exactly 4 digits ("from 4 through 4")
\(\d\) matches a period followed by a single digit


The replacement

Baroque 11

\1 is replaced by the 1st (\d\d) pattern matched

\(\d\d\)

A Solution For Problem 3 using Python (a system-level scripting)

Source text:

'ALG','1','15.00','Written problem','25 Sep 1986'
'ALG','2','15.00','Written problem','19 Oct 1986'

The Script — which replaces '25 Sep 1986' by '09-25-1986', etc.

```python
import re
import fileinput

monthNumber = {}  
monthNumber["Jan"] = "01"  
monthNumber["Feb"] = "02"  
monthNumber["Mar"] = "03"  
monthNumber["Apr"] = "04"  
monthNumber["May"] = "05"  
monthNumber["Jun"] = "06"  
monthNumber["Jul"] = "07"  
monthNumber["Aug"] = "08"  
monthNumber["Sep"] = "09"  
monthNumber["Oct"] = "10"  
monthNumber["Nov"] = "11"  
monthNumber["Dec"] = "12"

compiledPattern = re.compile( r'\^\(\d\d\)RC 741 20061112 \d\{4,4\}\(\d\d\)\d\$' )

for nextLineOfInput in fileinput.input():
    matchedObject = re.match( compiledPattern, nextLineOfInput )
    ( prefix, dayString, monthString, yearString, suffix ) = matchedObject.group( 1, 2, 3, 4, 5 )
    print prefix + monthNumber[monthString] + "+" + dayString + "+" + yearString + suffix
```

Which produces:

'ALG','1','15.00','Written problem','09-25-1986'
'ALG','2','15.00','Written problem','10-19-1986'
Hints For Using Regular Expressions

To build a complex pattern
• start simple
• add to pattern one small piece at a time
• test as you go

When doing replacements
• build and test the search pattern first
• then build the replacement

Save useful patterns in a text file

If something doesn’t seem to work, check the documentation
• details of regular expressions syntax vary from application to application
• eg MS Word uses non-standard syntax, and lacks some standard features (sigh...)

Useful Software for Experimenting With Regular Expressions

RegExplorer (Shareware for Mac & Windows, $12 US)

A Better Finder Rename (Shareware for Mac & Windows, $20 US)
Mac: http://www.publicspace.net/ABetterFinderRename/
A Better Finder Select ($10) is interesting, too.