

Regular Expressions



BIT Extended Education
Session 3, Part 1

<http://www.mcs.vuw.ac.nz/~donald/?PerlTalk>

Setup

`/vol/projects/bitee/init`

Regular Expressions

- Describe text to be matched
- Uses
 - Validate input data
 - Parsing files
 - Searching through files
 - Transforming data
- Supported by several Unix utilities
 - grep/egrep
 - sed
- Perl has regular expression support as part of the language syntax
- Several extended versions exist

Visual REGEXP

- Simple program to visually test regular expressions
- Check the "lineanchor" box
- Top text area contains the regular expression
- Bottom text area contains text to be matched
- "Go" button will attempt to match text with expression

grep

grep regular_expression list of files

- Takes standard input, or a set of files, and prints out each line which matches the given regular expression
- You may have to escape parts of the regular expression from the shell -- if you have problems, put apostrophes around the expression
- egrep supports extended regular expression syntax; use it if your expression is even remotely complicated

Matching literal text

Individual characters in a sequence will match that same sequence of characters. Characters which mean something special in regular expressions may be matched by prefixing them with a backslash (\).

sam

matches

Many **s**amples were contaminated
by bal**s**amic vinegar added by
overzealous **s**amurai.

Matching more than once

After a character or subexpression,

- ? matches 0 or 1 times
- * matches 0 or more times
- + matches 1 or more times
- {n} matches exactly n times
- {n,} matches n or more times
- {n,m} matches n-m times

`e?s{2,}`

matches

"Y**esssssss**, my preciou**sssssss**",
said Gollum menacingly.

Metacharacters

. any character

^ beginning of a line/string

\$ end of a line/string

\s whitespace

\b edge of a word *

\B something other than the edge of a word *

* unsupported by Visual REGEXP

^donald.*emacs
matches

```
donald 16786 0.0 1.5 18172 15816 pe SN 26Aug04 21:54.86 emacs gui.bsh
donald 15207 0.0 1.2 5632 12144 rc SN+ Sun05PM 0:10.71 kregexpeditor
donald 18407 0.0 0.0 732 4 rc IWNs 22Sep04 0:00.02 bash
donald 22959 0.0 1.1 10552 11608 q8 SN Wed05PM 2:34.28 emacs submit
```

Grouping, match "any of" (1)

Parts of an expression can be grouped together, using brackets

- Entire subexpressions are affected by the modifiers on the previous slide
- The | character allows expressions to match one subexpression or another

Expressions can match one of a set of characters

- [donal] will match any of the letters in my name
- [0-9] will match any digit
- [^] will match anything but a space

Grouping, match "any of" (2)

Match HTML tags with href or src attributes

```
<[a-z0-9]+\s+([\^>]+\s+|)(href|src)[\^>]*>
```

matches

```
<html>
```

```
<head><title>Hello, world!</title></head>
```

```
<body>
```

```
<h1><a href="http://www.microsoft.com/">Where  
do you want to go today?</a></h1>
```

```

```

```
</body>
```

```
</html>
```

Your tasks

Write regular expressions to match

- Email addresses (user@host.domain)
 - remember, domain names can only contain digits, letters, hyphens and dots
- URLs that are likely to refer to static files
 - protocol://host(:port)/path
 - JPEGs, GIFs, PDFs, HTML, SWF (flash), AVI, MPG, MOV files are likely to be static
- applet, embed and font tags in an HTML document
- all processes in the output of ps auxw which have used more than an hour of CPU time (the TIME column)

Example files to test your expressions against are in /vol/projects/bitee/regex/samples.

Now we've got that out of the way...

Perl



BIT Extended Education
Session 3, Part 2

<http://www.mcs.vuw.ac.nz/~donald/?PerlTalk>

Perl

- Portable Extraction and Reporting Language
- The "Swiss Army Chainsaw" of Unix
- An interpreted scripting language
- "There's more than one way to do it"
 - but many of them are wrong
- Easy to write unreadable code
- String processing is very easy
- Useful for task automation
- Useful for writing CGI scripts

What's it like?

- Scripting language -- not compiled
- Uses standard Unix scripting mechanism
 - file chmodded executable
 - first line `#!/usr/bin/perl`
- Commands separated by ; like Java
- Comment lines begin with #
- Variable names begin with a character specifying type

Data types

Variables

- Scalars
- @rrays (lists)
- %ashes (maps)
- and references, which act like scalars
- don't need to be declared, unless use strict; is turned on

Literal values

- "this is a string"
- ('this is', "several", 'in a', 'list')
- (these => "are", "in a" => 'hash')
- 42

Scalars

■ Strings

- 'Single quotes mean just a literal string'
- "Double quotes allow other \$variables to be included"
- "And control characters -- to start a new line after this one: \n"

```
$name = "Donald";
```

```
print "Hello, $name!\n";
```

■ Numeric values -- like Java

■ Operators

- Numeric comparison -- like Java (==, <=, >=, !=)
- String comparison -- ne, eq, gt, lt
- Logic -- like Java (&&, ||, !)

```
if ("perl" == "confusing") { print "...and?\n"; }
```

```
if ("logic" ne "completely lost")
```

```
{ print "thankfully.\n"; }
```

More scalars

■ String operators

- concatenation: .
- multiplication: x

```
$str = "One way to create a very long string "  
. "is to use the concatenation operator.";  
$str .= "But to make a really long string, "  
. "you can " . ("repeat " x 5) . "yourself.";
```

■ Numeric operators

- Just like Java: *, /, +, -, ++, --

Arrays and hashes

```
@courses = ("COMP301", "COMP311",  
            "COMP462");  
%office = ("Donald" => "CO231",  
          "Kirk" => "CO327",  
          "James" => "CO234",  
          "Neil" => "CO440");  
$office{"Pondy"} = "CO239";  
print "Donald's office is $office{\"Donald\"}.\n";  
$courses[1] = "INET101";  
push @courses, "COMP432";  
unshift @courses, "MATH114";
```

Conditionals

- if is mostly like Java, but the statements after it must be surrounded by braces
- unless means "if not"
- if and unless can be placed after a single statement

```
if ($logged_in == 0) {  
    exit(0);  
}
```

```
exit(0) unless ($logged_in);
```

Loops

- for is like Java
- foreach handles lists
- useful functions
 - keys returns a list of hash keys
 - sort sorts a list

```
for ($i = 10; $i >= 0; $i--) {  
  if($i) { print "$i...\n" }  
  else { print "Liftoff!\n" }  
}
```

```
foreach $i (sort keys %office) {  
  print "$i lives in $office{$i}\n";  
}
```

References

- can be put wherever you put a scalar
- new array: `$newarray = [1, 2];`
- new hash: `$newhash = { a => 12, b => 42 };`
- access elements: `$newarray->{"key"}`
- for functions that require a real array/hash:
 - `sort @$newarray`
 - `keys %$newhash`
- the ONLY way you can store a list inside another list

```
$arrayref = [];
```

```
foreach $i (sort keys %office) {  
  push @$arrayref, "$i lives in $office{$i}";  
}
```

Regular Expressions and Files

- `<FILEHANDLE>` returns a line from an open file
- `<>` reads from the files listed on the command line, and then standard input
- `"scalar" =~ /regex/` matches against a regular expression

```
$badtags = 0;
while($l=<>) {
    chomp $l; # remove trailing \n, if any
    $badtags++ if ($l =~ /<(applet|embed)/i)
}
print "$badtags bad tags found\n";
```

More regular expressions

- `$scalar =~ s/regex/replacement/` replaces matching text
- `$scalar =~ s/regex/ $1."ant".$2 /e` replaces with an expression
- The text matching (sub)(expressions) is stored in variables `$1,$2,...`

```
while($|=<>) {  
  chomp $|;  
  if ($| =~ /^[error\].+(File[^:]+): (V.*)$/i) {  
    push @errors, { error => $1, file => $2 };  
  }  
}
```

```
foreach $i (@errors) {  
  print "$i->{file}: $i->{error}\n";  
}
```

```
[Thu Oct 7 11:19:31 2004] [error] [client 127.0.0.1] File does not exist: /u/others/train21/apache/htdocs/error
```

```
[Thu Oct 7 10:51:03 2004] [error] [client 127.0.0.1] file permissions deny server execution: /u/others/train21/apache/cgi-bin/test
```

Your task

Write a perl script to parse a web server access log

- sample in /vol/projects/bitee/access_log

Display the page URL with the most accesses

Remember to chmod it executable, or it won't run at all:

chmod 0755 scriptname

Documentation

- in a terminal window, perldoc perlfunc or perldoc perl
- <http://www.mcs.vuw.ac.nz/cgi-bin/perldoc>
 - again, ask for the perlfunc or perl pages

Even More Perl



BIT Extended Education
Session 3, Part 2

<http://www.mcs.vuw.ac.nz/~donald/?PerlTalk>

Something I was reminded of

```
if ($something) {  
} elseif ($somethingelse) {  
} else {  
}
```

Packages

- Like Java classes
- `perldoc perltoot`
- CPAN (<http://search.cpan.org>)
 - Automated install
 - Look there first
- Once installed, `perldoc Package::Name`

Syntax

```
#!/usr/bin/perl  
use CGI;
```

```
$q = new CGI;
```

```
$shellotext = "hello, world!";
```

```
$shellotext = "hello, " . $q->param('name') . "!"  
    if (defined($q->param('name')));
```

```
print $q->header,  
    $q->start_html($shellotext),  
    $q->h1($shellotext),  
    $q->start_form,  
    "What's your name? ", $q->textfield('name'),  
    $q->end_form,  
    $q->end_html;
```

Tied hashes and DBM

- Wouldn't it be nice to use files on disk like hashes?
- tie can connect one to a hash

```
use NDBM_File;
```

```
use Fcntl; # for O_RDWR
```

```
$db = tie(%hash, 'SDBM_File', 'my_database',  
           O_RDWR|O_CREAT, 0600);
```

```
$hash{"blah"}++;
```

```
untie(%hash);
```

Other useful modules

- LWP (perldoc lwpcook)
- Storable
- MLDBM (combines Storeable and tie)
- MIME::Lite (send mail with attachments)
- DBI (database stuff)
- ...many more: <http://search.cpan.org/>

Your task

`http://localhost:3000/cgi-bin/script.pl`
will run a CGI script located in
`~/apache/cgi-bin/script.pl`

You must `chmod` it `0755` or apache won't run it.

`print STDERR "debug information\n"` will cause things to appear in
`~/apache/log/error_log`

Using the CGI module, write a simple poll script that has a set of radio buttons to choose an option to vote for, and uses a tied hash to store vote counts.

`http://www.mcs.vuw.ac.nz/cgi-bin/perl/doc?entry=CGI`