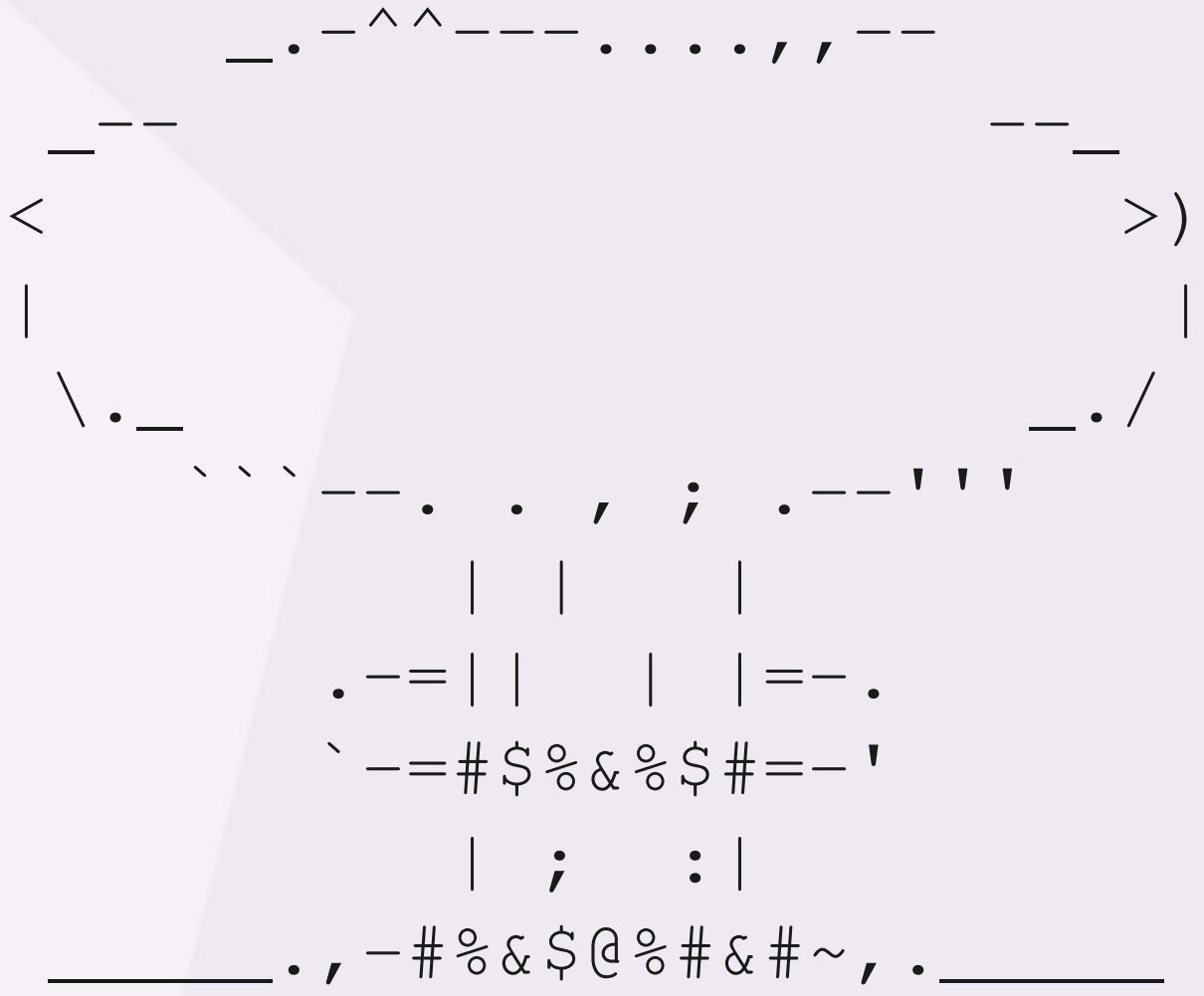
netdirect
open solutions

SERVER ISSUES KEEPING YOU UP?

CHOOSE NET DIRECT

Regular Expressions



It's a Text-Based World

- There is a lot of text in the world today.
  - Emails
  - Log files
  - Web content
- Lots of programs have text in and out
  - Program output
  - Internet protocols
  - Interactive programs

The need for pattern matching

- Search for things in email or documents
- Validation on program input
- Validate on web form submissions
- Reading text files programmatically
- Searching or filtering logs
- Filtering email
- Text processing

Enter Regular Expressions

- Invented by mathematician Stephen Cole Kleene in the 1950s.
- Put into practice by Ken Thompson in his QED editor, later Unix ed.
- Grep followed
- Awk, Perl and other programmatic support followed.
- Is available in countless programs and languages.
- Also called RegEx or RE. You might also see BRE (basic) and ERE (extended.)

Versions

- Dozens or more versions but only a few major ones.
  - Simple (deprecated)
  - Basic (use only if you have to)
  - Extended
  - Perl, PCRE (C library)
- Some utilities have odd extensions, most rely on libraries.

Literal Expressions

- Most characters match literally.
  - 'kwlug' matches “kwlug”
- They are case sensitive
  - 'kwlug' won't match “KWLUG”
- These are the simplest expressions
- Find an email address:
  - grep 'kwlug-disc@kwlug.org'

Metacharacters

- Metacharacters have special meaning and are the syntax of expressions
- Some represent characters, others repetition, others represent special conditions.

Meta	Meaning	Meta	Meaning
.	Any one character	[ ]	Ranges or classes of characters
*	0 or more of the previous pattern	( )	Grouping
+	1 or more of the previous pattern	|	Branching (or)
^	Beginning of line	\n	Back references
$	End of line	{n,m}	Repetition

Escapes

- To “invert” the meaning of a character use the backslash: \
- e.g. to match an actual square bracket use \[
- In Basic regex metacharacters need escaping e.g. \(\) and \{ \}, \+.
- In Extended regex ( ) and { } don't need escaping unless you want them to be literal.
- Try to use Extended RE when possible. My examples with use EREs.

Using Metacharacters

- '.' matches a single character any character.
- '..' matches any two characters.
- Common pattern is '.*'. It matches a string of any characters including a 0 length string.
- Another common one is '.+'. It matches a string of any characters 1 character or longer.
- In any case '.*' and '.+' doesn't mean the same character has to be repeated. Any sequence of characters will match.

Literals with Metacharacters

- '200.' would match “2000” through “2009” but also a hex number like “200a”, or just “200” followed by any character including space.
- ' *' (space, asterisk) would match any number of spaces include none.
- 'Jun.*kwlug-disc' match any text where “Jun” is followed by “kwlug-disc”.

Line Oriented

- In command line tools expressions are often line oriented.
- In programmatic tools expressions are often string oriented.
- Enter '^' and '$': beginning of line and end of the line or string.
- Matches the logical begin and end of a line/string but don't actually match any characters.

Using ^ and $

- '^The' matches a line that begins with “The”.
- '^The end$' matches an entire line, one that has only “The end”.
- ' +$' finds spaces at the end of lines.
- '^ +$' finds lines that only have spaces.
- '^$' find empty lines.

Useful simple examples

- grep '^Mar 2' /var/log/maillog
  - Show all mail messages for a single date
- ps -e | grep 'd$'
  - Show daemon processes (ending in d)
- grep '^Mar .*kwlug-disc' /var/log/maillog
  - Find all messages for kwlug-disc in March

Bracket expressions

- Uses square brackets [and]
- Contains a list of characters to match
- Matches only one character.
- Can use ranges e.g. 0-9 a-z A-Z
- First character can be special
  - - indicates literal dash
  - ^ indicates negation, not one of these chars.

Bracket examples

- '[a-zA-Z]' match only an alphabetic character.
- '[0-9a-zA-Z]' match only an alphanumeric char.
- '[0-9a-fA-F]' match only a hexadecimal char.
- '[-+.0-9()]' match a telephone number char.
- '[hc]at' match the word hat or cat.
- '[^"]' match every character except a quote.

Grouping/Atoms

- Atoms are a single unit of pattern, usually a single character
- The parentheses (and) (or \(and \) in Basic expressions) group small atoms together into a bigger atom:
- Useful where patterns repeat
  - e.g. '(bon)* matches bon or bonbon or bonbonbon or nothing.
- Also used in branching and back references.

Repetition

- The { } brackets can be used to specify the number a pattern repeats.
- In Basic regex we have to escape these (\) for them to be taken as metacharacters
- Examples
  - '.{80}' eighty characters
  - '.{60,80}' matches as few as 60 or as many as 80 characters.
  - e.g. '[-+. ()0-9]{7,16}' *could* match a phone number.

Branching (or)

- Branching provides alternative sub-patterns in a pattern, essentially a logical OR capability
  - 'a|b|c|d|e|f' matches one character of a through f
- To match longer patterns use brackets:
  - '\.(ca|com|edu|net|org)' matches global top level domains in a domain name or email address.
  - Means ca **or** com **or** edu **or** net **or** org

Syntax Chart

- Universal syntax

Description	Syntax
Any character	.
0 or more occurances	*
Beginning of line	^
End of line	$
Brackets	[...]

- Variations

Description	Basic	Extended	Perl
1 or more	\+	+	+
repitition	\{n,m\}	{n,m}	{n,m}
Grouping	\(...\)	(...)	(...)
Backreferences	\n	\n	$1

Classes and other syntax

- Word boundary: \< \> or \b
  - `echo cantilevers cant work | sed "s/\<cant\>/can't/g"`
  - `echo cantilevers cant work | sed "s/\bcant\b/can't/g"`
- Non-word boundary: \B
- [[:alpha:]] = [a-zA-Z]
- [[:space:]] = [\t\n\r] (\t = tab)
- Others: [[:alnum:]], [[:digit:]], [[:lower:]], [[:upper:]], and more

Greedy Matching

- Matches as much as possible. Early patterns get the most.
  - `echo goodbye cruel world | sed -r 's/(.*) .*/\1/'`
  - Results in “goodbye cruel”
- Perl has a direct way to prevent greedy matching: '?' Is the “lazy” quantifier:
  - '.*? .*'

Let's match an email address

- Start simple: .*@.* matches an email address, but it also matches other things as well:
  - 4 winter tires @ $99 each
- So lets get more complex: [a-z]*@[a-z.]*
  - Now it must be letters, at sign followed by letters and/or dots
  - Also matches:
    - @copyright
    - meet@
    - nick@night

Better email address pattern

- Okay, there must be at least one letter on each side: [a-z]+@[a-z.]+
  - Still doesn't force the use of a period.
- How about: [a-z]+@[a-z]+\.[a-z]+
  - Better, but it also matches: john@netdirect.canada
- Better sanity check: [a-z]+@[a-z]+\.[a-z]{2,6}
  - TLDs are 2 to 6 chars in length
  - But this won't match: john@office.netdirect.ca

Even Better Email Address Pattern

- We want to allow many domain name parts:
  - [a-z]+@([a-z]+\.)+[a-z]{2,6}
  - Way better, but what about other valid characters?
- Add in other valid characters like uppercase, hyphen, underscore, plus:
  - [-+_.a-zA-Z]+@([-a-zA-Z]+\.)+[a-zA-Z]{2,6}

Still not perfect

- Doesn't match IP address domain parts:
  - john@[216.16.235.8]
- Won't deal with:
  - John Van Ostrand <john@netdirect.ca>
- Doesn't work with foreign char sets,
- Also matches non-tlds like “linux@is.awesum”
- Go as simple or as complex as you need.

Exact TLD match

- [-+_.a-zA-Z]+@([-a-zA-Z]+\.)+(AC|AD|AE|AERO|AF|AG|AI|AL|AM|AN|AO|AQ|AR|ARPA|AS|AT|AU|For|AW|AX|AZ|BA|BB|BD|BE|BF|BG|BH|BI|BIZ|BJ|BM|BN|BO|BR|BS|BT|BV|BW|BY|BZ|CA|CAT|CC|CD|CF|CG|CH|CI|CK|CL|CM|CN|CO|COM|COOP|CR|CU|CV|CX|CY|CZ|DE|DJ|DK|DM|DO|DZ|EC|EDU|EE|EG|ER|ES|ET|EU|FI|FJ|FK|FM|FO|FR|GA|GB|GD|GE|GF|GG|GH|GI|GL|GM|GN|GOV|GP|GQ|GR|GS|GT|GU|GW|GY|HK|HM|HN|HR|HT|HU|ID|IE|IL|IM|IN|INFO|INT|IO|IQ|IR|IS|IT|JE|JM|JO|JOBS|JP|KE|KG|KH|KI|KM|KN|KR|KW|KY|KZ|LA|LB|LC|LI|LK|LR|LS|LT|LU|LV|LY|MA|MC|MD|MG|MH|MIL|MK|ML|MM|MN|MO|MOBI|MP|MQ|MR|MS|MT|MU|MUSEUM|MV|MW|MX|MY|MZ|NA|NAME|NC|NE|NET|NF|NG|NI|NL|NO|NP|NR|NU|NZ|OM|ORG|PA|PE|PF|PG|PH|PK|PL|PM|PN|PR|PRO|PS|PT|PW|PY|QA|RE|RO|RU|RW|SA|SB|SC|SD|SE|SG|SH|SI|SJ|SK|SL|SM|SN|SO|SR|ST|SU|SV|SY|SZ|TC|TD|TF|TG|TH|TJ|TK|TL|TM|TN|TO|TP|TR|TRAVEL|TT|TV|TW|TZ|UA|UG|UK|UM|US|UY|UZ|VA|VC|VE|VG|VI|VN|VU|WF|WS|YE|YT|YU|ZA|ZM|ZWac|ad|ae|aero|af|ag|ai|al|am|an|ao|aq|ar|arpa|as|at|au|for|aw|ax|az|ba|bb|bd|be|bf|bg|bh|bi|biz|bj|bm|bn|bo|br|bs|bt|bv|bw|by|bz|ca|cat|cc|cd|cf|cg|ch|ci|ck|cl|cm|cn|co|com|coop|cr|cu|cv|cx|cy|cz|de|dj|dk|dm|do|dz|ec|edu|ee|eg|er|es|et|eu|fi|fj|fk|fm|fo|fr|ga|gb|gd|ge|gf|gg|gh|gi|gl|gm|gn|gov|gp|gq|gr|gs|gt|gu|gw|gy|hk|hm|hn|hr|ht|hu|id|ie|il|im|in|info|int|io|iq|ir|is|it|je|jm|jo|jobs|jp|ke|kg|kh|ki|km|kn|kr|kw|ky|kz|la|lb|lc|li|lk|lr|ls|lt|lu|lv|ly|ma|mc|md|mg|mh|mil|mk|ml|mm|mn|mo|mobi|mp|mq|mr|ms|mt|mu|museum|mv|mw|mx|my|mz|na|name|nc|ne|net|nf|ng|ni|nl|no|np|nr|nu|nz|om|org|pa|pe|pf|pg|ph|pk|pl|pm|pn|pr|pro|ps|pt|pw|py|qa|re|ro|ru|rw|sa|sb|sc|sd|se|sg|sh|si|sj|sk|sl|sm|sn|so|sr|st|su|sv|sy|sz|tc|td|tf|tg|th|tj|tk|tl|tm|tn|to|tp|tr|travel|tt|tv|tw|tz|ua|ug|uk|um|us|uy|uz|va|vc|ve|vg|vi|vn|vu|wf|ws|ye|yt|yu|za|zm|zw)
- Shall we match IP address domain names now?

Tools

Command	RE Type
grep	basic
grep -E, egrep	extended
sed	basic
sed -r	extended
awk	extended
perl	Perl
ed	basic
vi	Basic extended???
less	extended

Using grep

- grep 'pattern' file
- grep 'pattern' < file
- cat file | grep 'pattern'
- Example: searching log files for “ssh”
  - grep ssh /var/log/secure
- Find ssh messages for Feb 1st
  - grep 'Feb 1.*ssh' /var/log/secure

Grep Tips

- -E : Use extended regex
- -P : Use perl regex
- -v : Invert - remove lines that match pattern
- -R or -r : Recursive, search a directory tree of files
- -l : list file names of files that match

Command Line Gotchas

- Metacharacters (*, ?, [,], {, }) can be confused with shell expansion characters.
- Pattern literals (', “, |, &, <, >, (,)) may be interpreted by the shell.
- Backslash may be interpreted by the shell.
- Use single quotes when possible or use double quotes and 'backslash' the problem characters “, *, $, \, etc.
- Shell syntax says single quote means no shell expansion. Double quote allows shell expansion.
- If you suspect a problem “echo” the expression to see how the shell handles it.

Examples of Problems

- Backslashes
  - grep “\$100”
- File expansion. If a file named “hello.to.the.world” existed
  - grep -E hello.*world
  - grep -E hel[a-z]*
- Brace expansion
  - grep -E hello{1,2}

Using sed

- Sed is a stream editor that modifies a text stream.
- A rich non-interactive editor
- I use it exclusively for text replacement:
  - sed “s/color/colour/g”
  - Replace “color” with “colour”
- Or deletion:
  - sed 's/:.*//'
  - Delete the colon and everything after.

Using Sed

- A trailing 'g' means global, repeat for all occurrences on a line.
- A trailing 'i' means ignore case
- Ranges can be used
  - sed '5,10s/^# //'
  - Comment out lines 5 through 10
- Ranges can be an REs
  - sed '/<title/,/<.title/s/Acme Inc/Net Direct Inc./g'
  - Replace “Acme Inc” with “Net Direct Inc.” between the “<title” line and the “</title” line

Back References

- Concept of “back references” allows you to re-arrange text or search for duplicate matches
- Flip words
  - echo hello world | sed -r 's/(.*) (.*)/\2 \1/'
- Find a palindrome:
  - grep -E '^([a-z])([a-z])([a-z])\2\1' /usr/share/dict/*
  - sed -rn '/^([a-z])([a-z])([a-z])\2\1$/p' /usr/share/dict/*

Handy Sed Tips

- -r: Use Extended regular expressions.
- -n: Don't print lines. By default sed prints non-matching lines and lines that have been matched. Use the p option in sed commands to print the line.
- -f <filename>: Obtain sed script from file name. Useful in avoiding shell interpretation.
- Slashes can be replaced with any other character. Useful if your pattern contains a /.
  - 's#1st#first#' is the same as 's/1st/first/`

Web Scraping

- Age of Persuasion (http://www.cbc.ca/ageofpersuasion/)
  - grep '<li class="module-list-item"'
  - sed 's/.*href="//;s/".*$//'
- This gets me each page that has the MP3 link
  - grep "<embed"
  - sed 's/.*external_url=//;s/["&].*//'
- Now I have URLs of all the episodes

AOP Web Scrape Script

```
#!/bin/bash
#       Download Age of Persuasion
wget -q -O - http://www.cbc.ca/ageofpersuasion/ |
        # filter out the list items
        grep "<li class=\"module-list-item\""  |
        # Extract the URLs
        sed 's/.*href="//;s/".*$//' |
        # For each URL
        while read url; do
                # Get the child page
                wget -q -O - "$url" |
                        # Get the "<embed>" object line
                        grep "<embed" |
                        # Get the MP3 URL
                        sed 's/.*external_url=//;s/["&].*//' |
                        # For each MP3
                        while read mp3; do
                                # Download the MP3
                                wget -nv "$mp3"
                        done
        done
```

RSS replacement

- wget -q -O - http://www.cbc.ca/technology/ |
  ```
  grep "class=\"headline\"" | sed -r
  's|.*href="([^"]*)">(.*)</a>.*|\2
  [http://www.cbc.ca\1]|'
  ```
- Or use sed -n option to eliminate grep
- ```
  wget -q -O - http://www.cbc.ca/technology/ |
  sed -rn
  's|.*class="headline".*href="([^"]*)">(.*)</a>.
  *|\2 [http://www.cbc.ca\1]|p'
  ```
  ```

What's my IP Address

- Find a service or web site that echoes your IP address. e.g. sendmail:
- telnet fife.sendmail.com 25

```
220 fife.sendmail.com ESMTP Sendmail Switch-3.4.2/Switch-3.4.2;
Fri, 22 Jan 2010 05:45:57 -0800
helo jvo
250 fife.sendmail.com Hello nic.NetDirect.CA [216.16.235.2],
pleased to meet you
quit
```

- (echo helo jvo; echo quit) | nc fife.sendmail.com 25 | sed -rn 's/250 .*\[([1-9.]+).*/\1/p'

AWK

- Regular Expressions are integral to the syntax
- Typical program structure:

```
#!/usr/bin/awk
BEGIN {
        actions
}
/regex/ {
        actions
}
END {
        actions
}
```

Counting line types

```
#!/usr/bin/awk -f

BEGIN {
        comment=0; blank=0; total=0
}
/^#/ {
        comment++
}
/^[ \t]*$/ {
        blank++
}
{
        total++
}
END {
        print "Comment lines: " comment
        print "Blank lines: " blank
        print "Config lines: " (total - comment - blank)
}
```

vim*

- Similar to sed
- Search:
  - :/pattern
- Replace
  - :start,stops/pattern/replace/
  - 1,$s/color/colour/g

Success with Regex

- Takes time
- Start with simple expressions
- Be prepared to climb the learning curve

Resource

- http://en.wikipedia.org/wiki/Regex
- man 7 regex
- http://www.hci-matters.com/blog/wp-content/uploads/2008/04/regex_chart_barnes.pdf
-

Contact Information

- http://www.netdirect.ca
- 866-883-1172
- info@netdirect.ca
- Questions??