USERBLUE

A SHARE FORUM FOR IBM USERS OF UNIX® SYSTEM TECHNOLOGIES

Regular Expressions
They're not just line noise

Harold Pritchett

The University of Georgia

Session 6505

Abstract

- The basis of all UNIX pattern matching utilities is the "regular expression." This session will provide an introduction to the regular expression for those who have not experienced it before, and a review for those who have.

The Speaker

Harold Pritchett

The University of Georgia

(706) 542-5110

harold@uga.edu

Disclaimer

Everybody has lawyers:

The ideas and concepts set forth in this presentation are solely those of the respective authors, and not of the companies and or vendors referenced within and these organizations do not endorse, guarantee, or otherwise certify any such ideas or concepts in application or usage. This material should be verified for applicability and correctness in each user environment. No warranty of any kind available.

Introduction

Who am I?

What makes me qualified to talk about this subject?

- 25 Years working with computers
- 10 Years experience with Unix
- Unix Security Administrator
- Security Incident Handling Team for UGA

Introduction

Regular expressions come in three types

- Shell - used by all common UNIX shells
- basic - used by grep and sed
- Advanced - used by egrep and awk

Introduction

Common utilities using Regular Expressions

- grep - simple find based upon contents
- egrep - complex find based upon contents
- sed - find and replace based upon contents
- awk - find and process

Common Characteristics

Work on text files only

Work on a line at a time

Use simi-filter file processing

Similar command line structure

Use selection/action paradigm

Use Regular Expression for selection

Sample Regular Expression

M[ou]'?am+[ae]r .*([AEae]l[-])? [GKQ]h?[aeu]+([dtz][dhz]?)+af[iy]

(We will show what this matches at the end of the talk)

String Matching

A Regular Expression is always a string matching mechanism.

The match proceeds left-to-right in simple comparison steps across both:

- The string being matched and
- The pattern defining the match

Characteristics

Matching proceeds left to right

The leftmost longest match is always made at each step

Matching is iterative. If a comparison step fails, the match backtracks if possible.

Common Features

Any character which is not a meta-character matches itself /abc/

This is UNIX -- all these operations are case sensitive /abc/ vs /ABC/

Common Features

Classes are described using meta-characters; the most common meta-characters are \ and [and]

Any character preceded by \ matches itself whether or not it is a meta-character /\[/

Common Features

Any string of characters in square brackets
matches exactly one of the enclosed characters;
commonly called a character class /[abcde]/

^ as the first character within [] means the
complement of the set of characters, not
including \n /[^abcde]/

Common Features

- within (i.e. not first) [] means a contiguous range of characters; this may not work if the processor is not an ASCII-native machine /[a-e]/

Three types of RE
(Shell, basic and advanced)

All of the common shells - sh, csh, ksh, bash, etc. - use the same type of REs for selecting filenames

grep and sed use the basic REs for selecting text lines

egrep and awk use advanced REs for selecting text lines

Shell Regular Expressions

Sometime referred to as “wild card” matching

Automatically anchored to both the beginning and end of the line

The first regular expression in the pattern must match the first character in the file name and the last regular expression in the pattern must match the last character in the file name

Shell Regular Expressions

The Bourne, Korn and C shells recognize a common set of metacharacters

? [ ] ! - *

Escaping in shell REs is rarely necessary, because file names rarely include Metacharacters.

Shell Regular Expressions

? - Matches any single character

* - Matches zero or more characters

[] - define a character class

! - Only within []

- - Only within []

Shell Regular Expressions

The **?** Character matches any single character

h?t

Matches

hat

hit

hot

hut

But also matches

hbt

hct

hdt

Shell Regular Expressions

[abcd]	Matches a single character from the set a, b, c, or d.
[a-d]	Exactly the same as above
[-ad]	Matches a single character from the set a, d, and hyphen.
[a\-d]	Exactly the same as above
[!abc]	Matches any regular character **EXCEPT** a, b, or c.

Shell Regular Expressions

Example:

h[aiou]t

Matches

hat

hit

hot

hut

But nothing else, expecially not haiout.

Shell Regular Expressions

The * character matches zero or more arbitrary characters.

h*t

Matches

ht

hit

height

hottest

And more, but does not match

mad_hatter

Shell Regular Expressions

To Match “mad_hatter” use something like:

*h*t*

This matches in 5 steps

```
    m a d _ h a t t e r
1   *--------
2           h
3             *--       (longest match)
4                 t
5                   *---
```

Basic RE Meta-characters

- ^ at the beginning of the RE, means match only at the beginning of the line
- $ at the end of the RE, means match only at the end of the line
- * means repeat the previous item an indefinite number of times, from 0 up
- . means any character except \n

Note the difference in behavior of "*" from the shell regular expression

Differences between Shell and General Regular Expressions

Shell	General	Meaning
?	.	Wildcard
*****	**.***	Zero or more wildcards
[ ]	**[ ]**	Character class
[!]	**[^]**	Reverse Character class
re	**^re$**	Fully anchored matching

BASIC RE Meta-Characters

\{n\}	matches exactly n occurrences of the preceding RE term; only in sed and grep
\{n,\}	matches at least n occurrences of the preceding RE term; only in sed and grep
\{n,m\}	matches between n and m occurrences of the preceding RE term; only in sed and grep

Basic RE Meta-characters

\< Matches at beginning of word

\> Matches at end of word

\(RE\) Matches the same as RE, but saves the result in one of nine substring registers. This result can later be used on the right hand side of a substitution using the special variable **\n**

Extended RE Meta-characters

| alternate choices at this location in the string **/ab|cd/**

() group REs for processing **/(ab|cd)ef/**

+ matches one or more of the preceding item **/[0-9]+/**

? matches 0 or 1 of the preceding item **/-?/**

Examples

From abc to line noise

/abcde/ -- matches the consecutive characters 'a', 'b', 'c', 'd', and 'e' in that order anywhere in a line

/[I-nI-N]/ -- matches any one of the letters which, by default, begin a FORTRAN integer variable

/[0-9][0-9]*/ -- matches any integer of one or more digits anywhere in the line

Examples

From abc to line noise

/^**abcde**/ -- matches the consecutive characters 'a', 'b', 'c', 'd', and 'e' in that order only at the beginning of a line

/^**abcde$**/ -- matches a line whose entire content is the consecutive letters 'a', 'b', 'c', 'd' and 'e'

/^**$**/ -- matches empty lines

Examples

From abc to line noise

/re[aei]d/ -- matches any one of the strings 'read', 'reed', or 'reid' anywhere in a line

/re[aei]?d/ -- adds 'red' to the allowed matches

/r.d/ -- matches 'r', any printable character, and 'd'

/r.*d/ -- matches 'r' followed by any number of characters followed by 'd'; dangerous since it matches 'read red reed, Reid' as a single item

Examples

From abc to line noise

/(\+|-)?[0-9]+(\.[0-9]\{0,8\})?\ ([eE](\+|-)?[0-9]\{1,3\})?/

matches an optionally signed number followed by an optional decimal part which may or may not have any digits but may have no more than 8 digits followed by an exponent part with an optional sign and at least 1 and no more than 3 digits

Important things
which will bite you

- Make sure that your RE will not match the null string (or any arbitrary string) Be careful with the use of the asterisk modifier (*)

- Don't forget to protect your scripts from the shell; almost all of the meta-characters for grep|egrep|sed scripts are also shell meta-characters and the shell **will** attempt to interpret them rather than pass them to the utility if you forget to quote them

The Answer

The example regular expression displayed at the beginning of this talk was taken from the test suite for the GNU grep program. It matches 32 different valid transliterations from Arabic of the name of the Libyan dictator.

M[ou]'?am+[ae]r .*([AEae]l[-])? [GKQ]h?[aeu]+([dtz][dhz]?)+af[iy]

The Answer

1) Muammar Qaddafi
2) Mo'ammar Gadhafi
3) Muammar Kaddafi
4) Muammar Qadhafi
5) Moammar El Kadhafi
6) Muammar Gadafi
7) Mu'ammar al-Qadafi
8) Moamer El Kazzafi
9) Moamar al-Gaddafi
10) Mu'ammar Al Qathafi
11) Muammar Al Qathafi
12) Mo'ammar el-Gadhafi
13) Moamar El Kadhafi
14) Muammar al-Qadhafi
15) Mu'ammar al-Qadhdhafi
16) Mu'ammar Qadafi
17) Moamar Gaddafi
18) Mu'ammar Qadhdhafi
19) Muammar Khaddafi
20) Muammar al-Khaddafi
21) Mu'amar al-Kadafi
22) Muammar Ghaddafy
23) Muammar Ghadafi
24) Muammar Ghaddafi
25) Muamar Kaddafi
26) Muammar Quathafi
27) Muammar Gheddafi
28) Muamar Al-Kaddafi
29) Moammar Khadafy
30) Moammar Qudhafi
31) Mu'ammar al-Qaddafi
32) Mulazim Awwal Mu'ammar Muhammad Abu Minyar al-Qadhafi

Questions?