CDOT
SICT AR Meeting Area
People

get involved with CDOT

as a Student
as an Open Source Community Member
as a Company

courses

BTC640
BTH740
BTP300
DPI908
DPS901
DPS905
DPS909
DPS911
DPS914
DPS915
DPS924
DPS931
EAC234
ECL500
GAM531
GAM666
GAM670
GPU610
LUX Program
MAP524
OOP344
OPS235
OPS245
OPS335
OPS345
OPS435
OPS445
OPS535
OPS635
OSD600
OSD700
OSL640
OSL740
OSL840

ULI101 Week 9

Contents [hide]

Regular Expressions [edit]

Define set of characters using a simple expression or a pattern. Used mainly for searching and/or replacing strings. Used by various Unix utilities such as:

- vi
- grep
- awk
- sed

Regular expressions match input within a line. Regular expressions are very different than shell meta-characters.

Literal Matching [edit]

Contains no special characters. Matches only itself: hence the term *literal matching*. Matches entire words or parts of it. Some examples of literal matching are:

String literal	Will match
`/disk/`	diskette, disk, disks
`/my book/`	my book, dummy book

> The slash character (`/`) at the start and end of the regular expression patterns used above and below are not part of the string pattern, rather the first slash is used to show where the string to match begins and the end slash is used to show where the string to match ends.

Regular Expression Delimiters [edit]

Real World Mozilla
RHT524
SBR600
SEC520
SPO600
SRT210
ULI101

course projects

Course Project List
Potential Course Projects
Contrib Opportunities

links

CDOT
Planet CDOT
FSOSS

Tools

What links here
Related changes
Upload file
Special pages
Printable version
Permanent link
Page information

Each regular expression should be delimited. This is particularly important to prevent the shell from interpreting special characters.

Delimiters mark beginning and end of the regular expression. Depending on a situation and utility used different characters can be used as a delimiter, for example in grep the delimiter is usually the double quote

Special Characters [edit]

Special characters and expressions can be used to build regular expressions for pattern matching. Standard special characters include:

`.`, `*`, `[]`, `^`, `$`

Depending on the utility and its version, some versions support standard regular expressions and some support extended regular expressions. Although some of them may look like shell expansion characters they usually mean something else. Whenever you wish to match a special character literally it must be quoted.

Period '`.`' [edit]

Period is the only wildcard in regular expressions (RE). The period character in a regular expression can be viewed as a placeholder (match) for any single character. Some examples of period being used in regular expressions:

Period (`.`) in RE	Will match
`/.nix/`	Unix, unix
`/leaf./`	leafs, leafy
`/a.t/`	a t, ant, act

Asterisk '`*`' [edit]

Represents zero or more occurrences of regular expression directly preceeding the asterisk (`*`). By itself, it does not match anything - it is NOT a wildcard. It is used in conjunction with literal matches, a period or other special characters. Some examples of asterisk in regular expressions:

Asterisk (`*`) in RE	Will match
`/car*/`	car, carpool, cart, caret
`/of.*ice/`	office, off ice

Square brackets '`[]`' [edit]

Enclose a character class or group, similar to the shell. Any single character within the brackets will be matched. Hyphen can be used for defining a range of characters. Most special characters lose their special meaning. The caret sign at the beginning of the list means exclusion (`[^a]` means *do not match a*). Some examples of square brackets in regular expressions are:

Square brackets (`[]`) in RE	Will match
`/practi[cs]ing/`	practicing and practising but not practicsing

`/file[12s]/`	file1, file2, and files but not file12s

Caret '`^`' [edit]

Matches strings at the beginning of the line (anchoring it). Special only if the beginning of the regular expression, otherwise means a literal match. Inside square brackets means character exclusion. Some examples of caret in regular expressions are:

Caret (`^`) in RE	Will match
`/^[0-9]/`	any line that begins with a digit
`/1[^0-3]/`	1a, 1., 145, but not 10 and 13

Dollar '`$`' [edit]

Matches strings at the end of the line, (anchoring matches to the end of the line). Examples of dollar in regular expressions are:

Dollar (`$`) in RE	Will match
`/A$/`	only match lines that end with capital letter A
`/50$/`	only lines ending in 50 like: 50, 150,

Grouping '`()`' and grouping with alteration '`(|)`' [edit]

Parentheses can be used to create bracketed regular expressions. The parentheses group the regular expression inside. The parentheses are not matched, only what is inside. Grouping offers alteration/choice represented by the pipe (|). When a grouped expression is followed by a quantifier such as the asterisk, the quantifier applies to the entire group Examples of grouping with and without alteration are:

Grouping (`()`) in RE	Will match
`/"(Mr\vertMrs) Smith"/`	"Mr Smith" and "Mrs Smith"
`/a(abc)*z/`	az, aabcz, aabcabcz

`grep` requires the `-E` option to enable grouping

Search and Replace in vi [edit]

Utilities such as vi are able to perform string substitution. Such substitution is done using regular expressions. You need to be careful when using non alpha-numeric characters - quote them if necessary. Examples of substitution syntax in vi:

```
:[address]s/original-string/replacement-string[/g]
```

where

`[address]`
: specifies a line range (optional). If address range is not given, then only current line is used for substitution

`[/g]`
: specifies global substitution on the line (optional). If `/g` is not given, then only the first

line occurence is substituted.

Example vi Substitution Ranges [edit]

In the following examples, substitute the first occurrence of 'a' with 'A'.

Where to replace 'a' with 'A'	vi command needed
on line 7 only	`:7s/a/A/`
on lines 7 to 10 (inclusive)	`:7,10s/a/A/`
in the entire document (range is %)	`:%s/a/A/`
from the beginning of document to current line (1,.)	`:1,.s/a/A/`
between current and next 5 lines (inclusive) (.,.+5)	`:.,.+5s/a/A/`

Categories: ULI101 | ULI101-2018