

Introduction to Regular Expressions

Demitri Muna
OSU

What Are Regular Expressions?

- Regular expressions are used for textual pattern matching.
- It's like half a computer language.
- Built into most modern programming languages and all good text editors
- Functionality provided on the command line though [grep](#).
- Can be linked as a library into any code, e.g. C.
- Best learned through examples.

grep

Open a terminal window in the same directory as the sample file `student_data.txt`.

Usage:

```
% grep <pattern> <files to search>
```

Simple searches:

```
% grep Junior student_data.txt
```

select all lines containing 'Junior'

```
% grep '^E' student_data.txt
```

^ = start of line: select all lines starting with 'E'

```
% grep 's$' student_data.txt
```

\$ = end of line: select all lines ending with 's'

```
% grep 'oom 3' student_data.txt
```

find all offices on third floor (note space requires quotes)

```
% grep 'oom [23]' student_data.txt
```

find all offices on second and third floors

Square Brackets

Square brackets match (i.e. find) a single character.

`[bc]at`

will match 'bat' and 'cat' but not 'mat' or 'hat'. You can place a range of letters or numbers inside the brackets.

`[b-k]at`

matches 'bat', 'cat', 'gat' but not 'mat', 'nat', 'pat'

`[3-7]5`

matches 35, 45, 55, 65, 75 but not 85, 95, etc.

`[a-z]`

matches any single lowercase letter

`[a-zA-Z]`

matches any single upper or lowercase letter

`[a-zA-Z]$`

matches any character that appears at the end of a line

`[^abc]`

matches any character *except* 'a', 'b', 'c' – (a '^' anywhere else in brackets just matches a caret)

On the command line:

`% ls -d .[a-zA-Z]*`

* = any length of characters : list any hidden (starting with a '.') file or directory beginning with a letter

Shortcuts

These are shortcuts that define frequently used patterns.

Shortcut	Matches
<code>\s</code>	any whitespace, e.g. space, tab, end of line (<code>\n</code> , <code>\r</code>), etc.
<code>\S</code>	any character <i>except</i> whitespace
<code>\d</code>	any digit : <code>[0-9]</code>
<code>\D</code>	any character <i>excluding</i> numbers
<code>\w</code>	effectively, a word : <code>[0-9a-zA-Z]</code>
<code>\W</code>	not a word : <code>[^0-9a-zA-Z]</code> (i.e. inverse of <code>\w</code>)

Grouping

You can identify groups of characters by using parentheses. This is useful for extracting text that matches a pattern or search and replace operations.

Example:

Swap the columns

'first_name' and 'last_name'.

```
first_name|last_name|city|supervisors|club|status
Caral|Rogers|New Britain|Tennant/Room 101|Sophomore|Che
Ori|Mejia|Lakeland||Senior|Debate
Leandra|Stevens|Rockford||Freshman|
Danielle|Moody|Oro Valley|Baker/Room 315, Eccleston/Ro
Josiah|Barber|Rancho Cordova||Sophomore|
Wing|Gordon|Reedsport|Baker/Room 315|Freshman|Rugby, C
```

Search for:

`^ ([A-Za-z_]+) \ | ([A-Za-z_]+) \ |`

start of line

any upper or
lowercase letter or
underscore

one or more
characters of the set

Replace with:

`\2 | \1 |`

second set of
parentheses

first set of parentheses

Note: what represents the groups differs between Python, Perl, text editors, etc.

Grouping

Grouping can also let you specify more complex queries:

`([ea][^r])`

Matches two characters beginning with either 'e' or 'a',
but *not* when followed by 'r'.

Imagine data files that look like this,
and you want to extract the
information from the filename.



id mjd version code

```
data-7542-55726-01f.par  
data-5468-55777-02f.par  
data-9875-55728-01x.par
```

data - 7542 - 55726 - 0 1 f . par

`([a-z]+)-([0-9]+)-([0-9]+)-([0-9][0-9])[a-z].par`

between 1 and 7
lowercase letters

exactly 4 digits

exactly 5 digits

two or three digits

single letter

`([a-z]{1,7})-([0-9]{4})-([0-9]{5})-([0-9]{2,3})[a-z]\.par`

'.' actually matches any character, so it should
be escaped with a backslash to match a period

Special Characters

Character	Meaning
<code>\n, \t, \r</code>	new line, tab, linefeed
<code>.</code>	any single character except a line break (<code>\n</code> , <code>\r</code>)
<code>?</code>	last item optional, e.g. <code>[a-z][0-9]?</code> matches 'a9' and 'x'
<code>+</code>	one or more of the last item, e.g. <code>[a-z]+</code> one or more lowercase letters
<code>\</code>	escape character, e.g. <code>\+</code> is an actual plus
<code>^</code>	start of line
<code>\$</code>	end of line

Example: `^.+` matches any full line.

Command Line Tricks

Useful for deleting certain ranges of files:

```
% rm file-2011-0[167]-*.txt
```

Deletes files from January, June, and July.

```
% rm */.[a-zA-Z0-9]*
```

Deletes hidden files in all subdirectories.

RegExp in Python

The `match` method searches from the *beginning* of the string.

```
import re          Import the regular expression module.

d = "The date is Jan-04-2011 today."

m = re.match("."+([A-Za-z]{3})\-([0-9]+)\-([0-9]+).+", d)

assert m is not None, "The pattern was not matched."

month = m.group(1)
day = m.group(2)
year = m.group(3)

print m.group(0)
print m.groups()
```

Three groups of parentheses for three matches, starting from index 1.

index 0 returns the original string

returns an array of all the grouped items

```
m = re.match("Jan", d)
m = re.match("The", d)
```

m is None (doesn't match start of line)

```
# typical usage
if (re.match(...)):
    # do stuff
```

If the string does not match the regular expression, match returns `None`.

The `search` method works in exactly the same way, but searches anywhere in the string (not strictly from the start).