

# Regular Expressions and Command-line Text Searching

Michael T. Goodrich  
University of California, Irvine

# Regular Expressions

- Regular expressions are an algebraic way of describing text patterns.
- Regular expressions support powerful searches.



# Basic Definition of Regular Expressions

- A **regular expression** (RE) defines a **language**, that is, a set of character strings
- A **single character** matches just that character.
  - RE  $a$  matches the string “a”
- **Concatenation**: concatenating two RE's  $x$  and  $y$  as  $xy$  matches any string  $\alpha\beta$ , where  $x$  matches  $\alpha$  and  $y$  matches  $\beta$ .
  - RE  $ab$  matches the string “ab”
- **Or**: joining two RE's  $x$  and  $y$  with the symbol  $|$  as  $x|y$  matches any string that matches  $x$  or  $y$ .
  - RE  $ab|cd$  matches the string “ab” or the string “cd”
  - Order: first Concatenation then Or

# Basic Definition of Regular Expressions

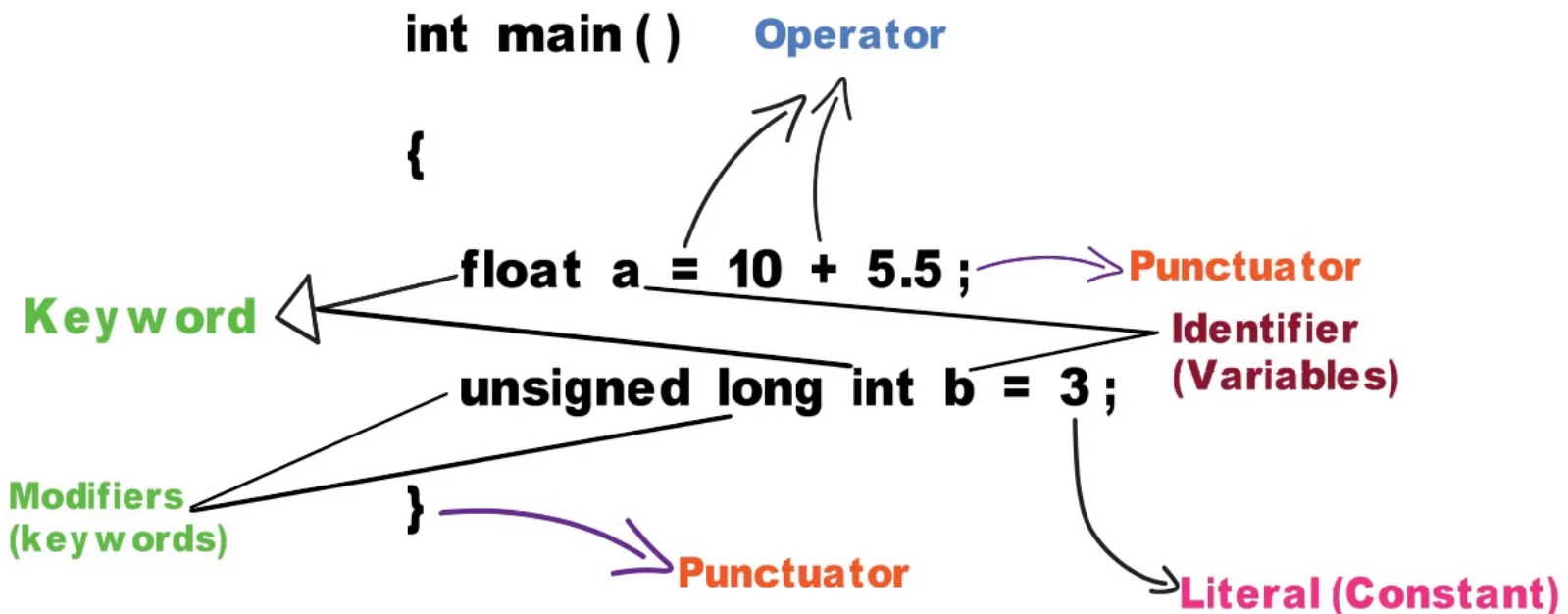
- **Parentheses:** joining an RE with parentheses matches the RE inside the parentheses and allows for grouping in the usual way.
  - RE  $a(a|bc)$  matches the strings “aa” and “abc”.
- **Closure:** given RE  $x$ , the RE  $x^*$  matches 0 or more repetitions of strings that match the RE  $x$ .
  - RE “ $ab^*$ ” matches strings “a”, “ab”, “abb”, “abbb”, ...
  - RE “ $a(a|b)^*$ ” matches any string of a’s and b’s that begins with an a.

## Some More Examples

- $(aa)^*$  matches any string consisting of an even number of a's.
- $(a|b)^*$  matches all strings of a's and b's (including the null string).
- $(a|b)^*aaa(a|b)^*$  matches any string of a's and b's containing three a's in a row.
- $(g|G)oodrich$  matches my last name with the first letter capitalized or not.
- $(g|G)(o|O)(o|O)(d|D)(r|R)(i|I)(c|C)(h|H)$  matches my last name ignoring case

# RE's are Surprisingly Powerful

- The tokens of every programming language are defined as regular expressions.



# Extended Regular Expressions

- **Extended regular expressions** provide additional syntax for defining regular expressions.
  - Every extended regular expression has an equivalent basic regular expression that defines the same language.
  - Extended regular expressions just give us a shorter way of defining regular expressions, for the sake of convenience.

Password

Minimum 8 chars

At least-one small letter [a-z]

At least-one upper letter [A-Z]

At least-one digit [0-9]

At least-one special char [~!@#\$%^&\*(...)]

# Extended Regular Expressions (egrep)

- A **bracket expression** is a list of characters enclosed by [ and ].
  - It matches any single character in that list
  - If the first character of the list is the caret ^ then it matches any character not in the list.
  - For example, the regular expression [0123456789] matches any single digit and [^0123456789] matches any single character that is not a digit.
  - Within a bracket expression, a **range** consists of two characters separated by a hyphen. It matches any single character between the two characters, inclusive, using the character set. For example, in the default C locale, [a-d] is equivalent to [abcd].



# Character Classes

- `[:alnum:]` matches alphanumeric characters. In the C locale, this is the same as `'[0-9A-Za-z]'`.
- `[:alpha:]` matches alphabet characters. In the C locale, this is the same as `'[A-Za-z]'`.
- `[:blank:]` matches blank characters space and tab.
- `[:cntrl:]` matches control characters.
- `[:digit:]` matches digits: 0 1 2 3 4 5 6 7 8 9.
- `[:graph:]` matches graphical characters: `[:alnum:]` and `[:punct:]`.

# Character Classes (continued)

- `[:lower:]` matches lower-case letters.
- `[:print:]` matches printable characters: `[:alnum:]`, `[:punct:]`, and space.
- `[:punct:]` matches punctuation characters; in the C locale, this is `! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~`.
- `[:space:]` matches space characters: in the 'C' locale, this is tab, newline, vertical tab, form feed, carriage return, and space.
- `[:upper:]` matches upper-case letters.
- `[:xdigit:]` matches hexadecimal digits: `0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f`.

# Other Matching Symbols

- The dot `.` symbol matches any character.
- The caret `^` and the dollar sign `$` are meta-characters that respectively match the empty string at the beginning and end of a line.
- The symbols `\<` and `\>` respectively match the empty string at the beginning or end of a word.
- The symbol `\w` is a synonym for `[_[:alnum:]]` and `\W` is a synonym for `[^_[:alnum:]]`.
- To match a special symbol as itself, you sometimes have to precede it with backslash `\`, e.g., `\$` matches a dollar sign. `\n` matches a newline (if used with `-z` option in `grep`).

# Repetition

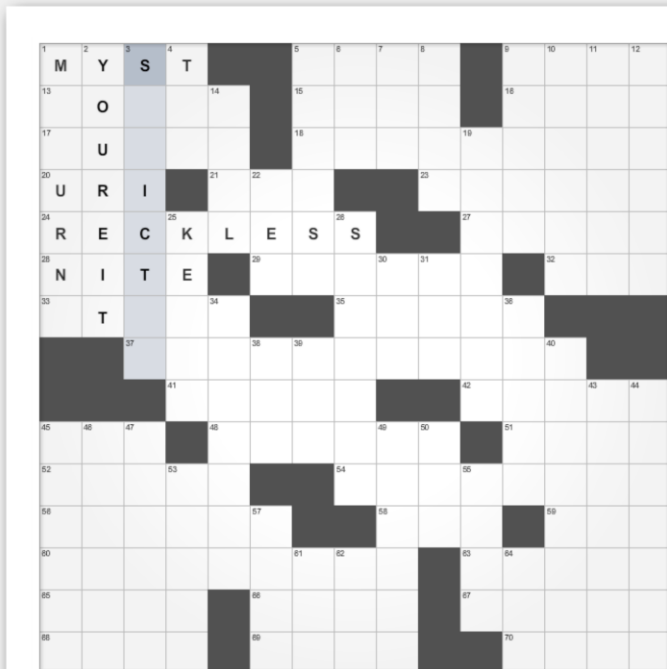
- An RE may be followed by a repetition operator:
  - ? The preceding item is optional and matched at most once.
  - \* The preceding item will be matched zero or more times.
  - + The preceding item will be matched one or more times.
  - {n} The preceding item is matched exactly n times.
  - {n,} The preceding item is matched n or more times.
  - {,m} The preceding item is matched at most m times.
  - {n,m} The preceding item is matched at least n times, but not more than m times.

# Some Examples

regular expression	matches	does not match
<code>.*SPB.*</code> <i>(substring search)</i>	RASPBERRY CRISPBREAD	SUBSPACE SUBSPECIES
<code>[0-9]{3}-[0-9]{2}-[0-9]{4}</code> <i>(U. S. Social Security numbers)</i>	166-11-4433 166-45-1111	11-55555555 8675309
<code>[a-z]+@([a-z]+\.)+(edu com)</code> <i>(simplified email addresses)</i>	wayne@princeton.edu rs@princeton.edu	spam@nowhere

# Another Example

- Unix/Linux/MacOS has a list of words in `/usr/share/dict` or `/usr/local/dict`. It is useful for doing regular expression word searches.



```
% more words.txt
```

```
a
aback
abacus
abalone
abandon
...
```

dictionary  
(standard in Unix)  
also on booksite

```
% grep "s..ict.." words.txt
constrictor
stricter
stricture
```

# Difference Between grep and egrep

- The command grep requires all the meta characters to be escaped with a backslash \ to have their special meaning.
  - In egrep, +, ?, |, (, and ), treated as meta characters, whereas in grep, they are treated as a pattern instead of meta characters.
  - By including backslash \ followed by meta character lets grep treat it as meta characters like \?, \+, \{, \|, \[, and \).
  - grep -E is the same as egrep

# Extended Example: Wordle

- Every day, a five-letter word is chosen which players aim to guess within six tries. After every guess, each letter is marked as either green, yellow or gray: green indicates that letter is correct and in the correct position, yellow means it is in the answer but not in the right position, while gray indicates it is not in the answer at all

A	R	I	S	E
R	O	U	T	E
R	U	L	E	S
R	E	B	U	S



# Game 285

S	P	E	L	T

```
% egrep "^[^spet]{3}l[^spet]$" words | wc
```

```
211  211 1266
```

# Game 285

S	P	E	L	T
B	U	I	L	D

akala	Dafila	manly
Alala	fally	Marla
alala	foaly	marly
amala	folly	Molly
amylo	fonly	molly
Calla	Galla	myall
callo	galla	nonly
Carlo	gally	oolly
chalk	Goala	ovolo
Chola	golly	oxfly
chola	holla	Paola
Cholo	hollo	Polly
coaly	Holly	rally
Colla	holly	Rollo
colly	hooly	Sally
cooly	jagla	wally
coyly	jolly	wanly
	joola	warly
	jowly	whalm
	knoll	whaly
	koala	wryly
	laxly	yalla
	lolly	yarly
	lowly	



% egrep "^[^spetbuid]{3}l[^spetbuid]\$" words | wc

64 64 384

# Game 285

S	P	E	L	T
B	U	I	L	D
C	O	O	L	Y



foaly  
folly  
fonly  
golly  
Holly  
holly  
jolly  
jowly  
lolly  
lowly  
Molly  
molly  
nonly  
Polly

```
% egrep "^[^spetbuidco]o[^spetbuidco]ly$" words | wc
```

14 14 84

# Game 285

S	P	E	L	T
B	U	I	L	D
C	O	O	L	Y
H	O	L	L	Y

```
% egrep "^lo[^spetbuidcohl]ly$" words
lowly
```

# Game 285

S	P	E	L	T
B	U	I	L	D
C	O	O	L	Y
H	O	L	L	Y
L	O	W	L	Y

# Difference Between grep and fgrep

- fgrep only does exact matching (same as grep -F):

```
% fgrep shaken /usr/share/dict/words
```

shaken

shakenly

unshaken

unshakenly

unshakenness



# Useful grep Options

♥ grep ♥

JULIA EVANS  
@b0rk

grep lets you search files for text

\$ grep bananas foo.txt

Here are some of my favourite grep command line arguments!



Use if you want regexps like ".+" to work. otherwise you need to use ".\+"



recursive! Search all the files in a directory.



invert match: find all lines that don't match



only print the matching part of the line (not the whole line)



case insensitive



only show the filenames of the files that matched



search binaries: treat binary data like it's text instead of ignoring it!



Show context for your search.



\$ grep -A 3 foo



will show 3 lines of context after a match



don't treat the match string as a regex  
eg \$ grep -F ...

grep alternatives



(better for searching code!)

# Useful Linux/Unix/Cygwin Commands

- wc
- cat
- diff
- find
- xargs
- grep
- sort
- uniq
- shasum
- sed
- strings
- join
- cut
- comm
- split
- paste
- tr





## WC


- Word count – prints number of lines, words, and bytes in a file (or from standard input).
- Useful options:
  - c The number of bytes in each input file is written to the standard output.
  - l The number of lines in each input file is written to the standard output.
  - m The number of characters in each input file is written to the standard output.
  - w The number of words in each input file is written to the standard output.

- Example:

```
% wc /usr/share/dict/words
```

```
235886 235886 2493109 /usr/share/dict/words
```

**Implies average number of letters in an English word is over 9 (wc counts newline characters)**



# cat

- Concatenate files and print to standard output.

- Example:

```
% fgrep dogs /usr/share/dict/words > dog-words.txt
```

```
% fgrep cats /usr/share/dict/words > cat-words.txt
```

```
% cat dog-words.txt cat-words.txt
```

dogs

dogship

dogshore

dogskin

dogsleep

dogstone

cat skin

catstep

catstick

catstitch

catstitcher

catstone

catsup

# diff

- Compare the differences between files.
- Options:
  - i Ignore case differences in file contents.
  - q Output only whether files differ.
  - y Output in two columns.
  - w Ignore all white space.
- Example:

```
% diff dogs.txt cats.txt
3c3
< dogs are great
---
> cats are fine
```

# find

- The find utility recursively descends the directory tree for each path listed, evaluating an expression.
- Options (too many to list – see “man find”)
- Example:

```
% find . -name "*txt"
```

```
./cats.txt
```

```
./dir2/Time fish.txt
```

```
./dir2/Horse fire.txt
```

```
./dir2/Dog breath.txt
```

```
./dir2/Pig breath.txt
```

```
./dogs.txt
```

```
./files2.txt
```

```
./dir1/Time fish.txt
```

```
./dir1/Horse fire.txt
```

```
./dir1/Dog breath.txt
```

```
./dir1/Pig breath.txt
```

```
./files1.txt
```

# xargs [utility [arguments]]

- xargs takes any arguments specified on the command line and gives them to utility upon each invocation. Example:

```
% find . -type f -print0 | xargs -0 wc
 3          9      44 ./cats.txt
 0         10    6148 ./DS_Store
 1          2       9 ./dir2/Time fish.txt
 1          2       9 ./dir2/Horse fire.txt
 2          3      15 ./dir2/Dog breath.txt
 2          3      14 ./dir2/Pig breath.txt
 3          9      45 ./dogs.txt
 4         12     263 ./files2.txt
 1          2       9 ./dir1/Time fish.txt
 1          2       9 ./dir1/Horse fire.txt
 1          1       6 ./dir1/Dog breath.txt
 1          1       5 ./dir1/Pig breath.txt
 4         14     257 ./files1.txt
24         70    6833 total
```

# sort

- sort sorts lines in a file by a specified column (1<sup>st</sup> by default)

```
% cat cats.txt
Linux Tutorials
Vim Editor
Sed Scripting
Awk Scripting
Bash Shell Scripting
Nagios Monitoring
OpenSSH
IPTables Firewall
Apache Web Server
MySQL Database
Perl Programming
Google Tutorials
Ubuntu Tutorials
PostgreSQL DB
Hello World Examples
C Programming
```

```
% sort cats.txt
Apache Web Server
Awk Scripting
Bash Shell Scripting
C Programming
Google Tutorials
Hello World Examples
IPTables Firewall
Linux Tutorials
MySQL Database
Nagios Monitoring
OpenSSH
Perl Programming
PostgreSQL DB
Sed Scripting
Ubuntu Tutorials
Vim Editor
```

# uniq

- The uniq utility reads a specified input\_file comparing adjacent lines, and writes a copy of each unique input line to the output\_file.
  - d Only output lines that are repeated in the input.
  - u Only output lines that are not repeated in the input.

```
% cat example.txt
cats
cats
dogs
pigs
rats
% uniq -d example.txt
cats
```

```
% uniq example.txt
cats
dogs
pigs
rats
% uniq -u example.txt
dogs
pigs
rats
```

# sed

- The sed utility reads the specified files, or the standard input if no files are specified, modifying the input as specified by a list of commands. The input is then written to the standard output.

- Example:

```
% sed "s/a/o/" example.txt
```

cots

cots

dogs

pigs

rots



# shasum

- Print or check SHA checksums

- Example:

```
% find . -type f -print0 | xargs -0 shasum
```

```
d62bc08671a4fc867fbb31e9d70cd0adc59e3cb9 ./dir2/Time fish.txt
55999560dc4321dc4650d2db7cc8a2c3eca1d7e4 ./dir2/Horse fire.txt
64d8b7e11ec7a7d7104de99d78b0c7054e93200a ./dir2/Dog breath.txt
775a42427b9062ea53595d25412eca93c48716f3 ./dir2/Pig breath.txt
d62bc08671a4fc867fbb31e9d70cd0adc59e3cb9 ./dir1/Time fish.txt
55999560dc4321dc4650d2db7cc8a2c3eca1d7e4 ./dir1/Horse fire.txt
b32f0990e2d7c9103a877f90677b88ab3dd03098 ./dir1/Dog breath.txt
cbcee2e6d3319eef25eff9325f8ece5674c1f154 ./dir1/Pig breath.txt
```

# join

- The join utility performs an “equality join” on the specified files and writes the result to the standard output.
- The “join field” is the field in each file by which the files are compared.
- The first field in each line is used by default.
- There is one line in the output for each pair of lines in file1 and file2 which have identical join fields.
- Each output line consists of the join field, the remaining fields from file1 and then the remaining fields from file2.

# Extended Example - Deduplication

```
% find ./dir1 -type f -print0 | xargs -0 shasum | sed "s/ /:/" | sort > files1.txt
% cat files1.txt
55999560dc4321dc4650d2db7cc8a2c3eca1d7e4:./dir1/Horse fire.txt
b32f0990e2d7c9103a877f90677b88ab3dd03098:./dir1/Dog breath.txt
cbcee2e6d3319eef25eff9325f8ece5674c1f154:./dir1/Pig breath.txt
d62bc08671a4fc867fbb31e9d70cd0adc59e3cb9:./dir1/Time fish.txt
% find ./dir2 -type f -print0 | xargs -0 shasum | sed "s/ /:/" | sort > files2.txt
% cat files2.txt
55999560dc4321dc4650d2db7cc8a2c3eca1d7e4:./dir2/Horse fire.txt
64d8b7e11ec7a7d7104de99d78b0c7054e93200a:./dir2/Dog breath.txt
775a42427b9062ea53595d25412eca93c48716f3:./dir2/Pig breath.txt
d62bc08671a4fc867fbb31e9d70cd0adc59e3cb9:./dir2/Time fish.txt
% join -t : -o 1.2 files1.txt files2.txt > duplicates.txt
% cat duplicates.txt
./dir1/Horse fire.txt
./dir1/Time fish.txt
```