Cleaning up data with Regular Expressions

Carleton University data boot camp - June 22, 2014
Glen McGregor
sushiboy21@gmail.com

The most effective way to clean up “dirty” or poorly formatted data is using a powerful set of search-and-replace commands called Regular Expressions. Also called RegEx, these functions are supported by text editors that have long been used by computer programmers and should be in every data journalists tool kit.

On Windows, try the open-source Notepad++. On a Mac, and in this tutorial, we’ll use TextWrangler.

We’ll work with a file of poorly formatted names and addresses that we’ll pull off the web from Canada411.com.

Enter Williams in the Name search window and Ontario in the Province search, the hit search. Now let’s hack the URL to increase the number of results to 100 but changing the pgLen option in the URL.

http://www.canada411.ca/search/si/1/Williams/Ontario/?pgLen=100

Highlight the list of names with your mouse, copy, then paste it all into a TextWrangler.

What a mess.

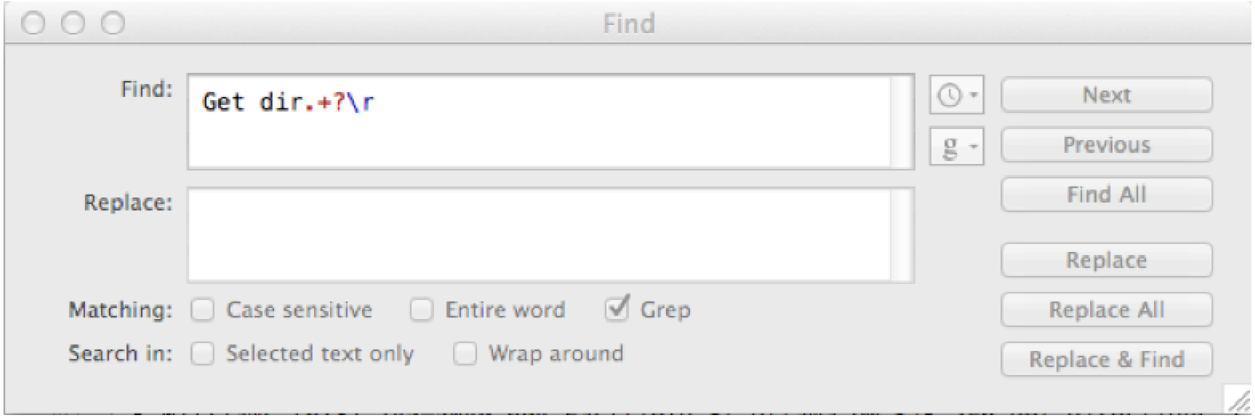
There are no tabs between fields and there’s a bit of garbage at the end of each line that we’d like to remove.

Let’s run a few Regular Expressions on the data, then start cleaning it up.

1. Start by downloading and installing TextWrangler from http://www.barebones.com/products/textwrangler/download.html, then open the program
2. Copy the data above COMMAND-C and paste it into TextWrangler COMMAND-V.
3. By default, TextWrangler will only search-and-replace data that comes after the cursor, so move the cursor to the top of the file.
4. We’ll start by getting rid of the junk words “Get directions →” Hit COMMAND-F to open the Find dialog box.
5. Make sure the checkbox for GREP is checked.
6. In the Find box, type **Get dir.+?\r .** This will look for the phrase **Get dir** and the any

other characters (designed by the **.** wildcard) that are repeated multiple times (the plus sign **+**). We keep the expression from being “greedy” but limited the selection (indicated by the **?**) when it reaches the end of the line (indicated by **\r**).

7. Because we’re selecting everything including the linebreak (indicated with \r) we need to put the line break back in. In the Replace window, type **\r**
8. The Find dialog box should look like this:



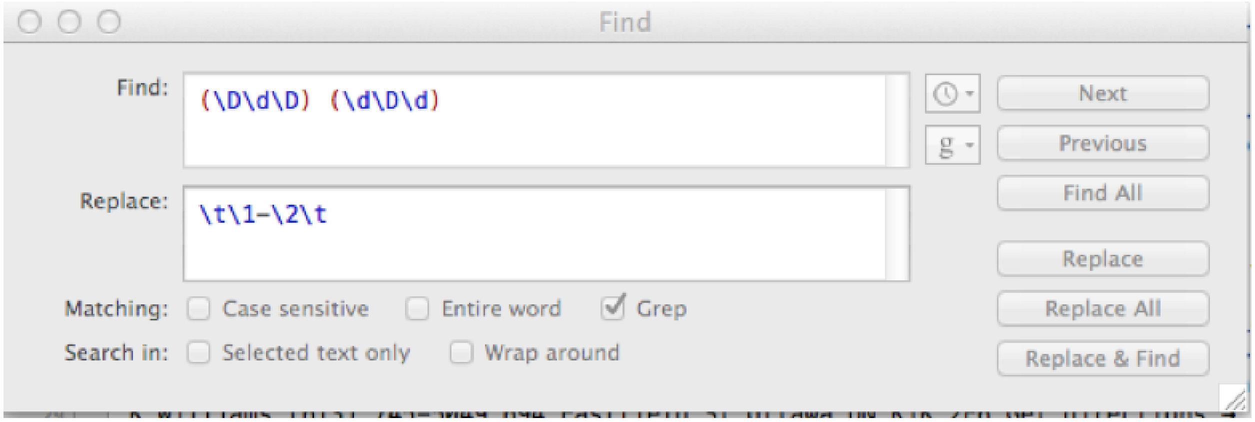
Find: Get dir.+?\r

Replace:

Matching: ☐ Case sensitive ☐ Entire word ☑ Grep

Search in: ☐ Selected text only ☐ Wrap around

9. You’ll notice the special characters **.+?** used by Regular Expessions are highlighted in red. Click Replace All. All instances of the words “Get directions” and the arrow sign after it should now disappear.
10. Next, let’s puts tabs around the postal codes so they will appear in their own column when we past the data in Excel.
11. We could use the Regular Expression **[A-Z][0-9][A-Z] [0-9][A-Z][0-9]** as our Find phrase. But Regular Expressions have some shortcuts that can be used to save a bit of typing. The expression **\d** will match all numbers while **\D** will match letters. So to find a postal code, our expression will be **\D\d\D \d\D\d .** Don’t forget to include the space.
12. We want to backreference the postal codes when we replace, so we’ll put round brackets around each part of the postal code in the Find dialog. It should look like this **(\D\d\D) (\d\D\d)**
13. In our replace box, we’ll put in a tab **\t**, then recall the first backreference **\1**, then for fun put in a hyphen **-** , then call the second backreference **\2** and then another tab **\t**. All together, it should look like this **\t\1-\2\t** and the dialog box should look like this:



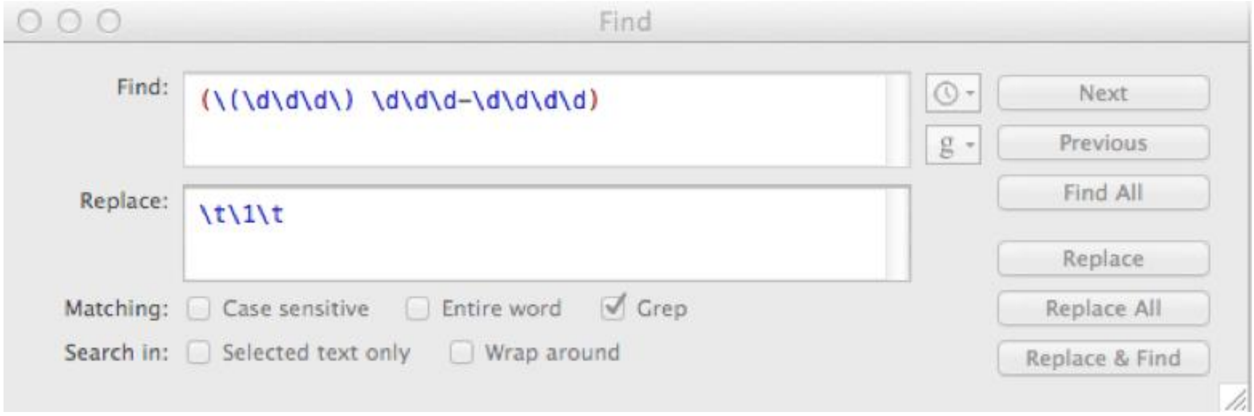
Find: `(\D\d\D) (\d\D\d)`

Replace: `\t\1-\2\t`

Matching: □ Case sensitive □ Entire word ☑ Grep

Search in: □ Selected text only □ Wrap around

14. Now click Replace All. Viola, all our postal codes are offset by tabs and we've put a hyphen in all the spaces between them.
15. Try putting in tabs on either side of the telephone number. It follows the pattern of three digits in round brackets**, (\d\d\d),** a space, three more digits **\d\d\d**, a hyphen, then four digits, **\d\d\d\d**. One problem here: the round bracket has a special meaning in Regular Expressions, for storing backreferences. If we want the find a real round bracket, we need to "escape" it using a backslash **** before it. So the expression that will find phone numbers is **\(\d\d\d\) \d\d\d-\d\d\d\d**
16. We also want to backreference the entire phone number so we need to put round brackets around it. Hit COMMAND-F for the Find dialog box and type into the Find window this expression: **(\(\d\d\d\) \d\d\d-\d\d\d\d).**
17. We'll replace this with a tab **\t**, the original phone number, and another tab **\t**. So the dialog box should look like this**:**



Find: `(\(\d\d\d\) \d\d\d-\d\d\d\d)`

Replace: `\t\1\t`

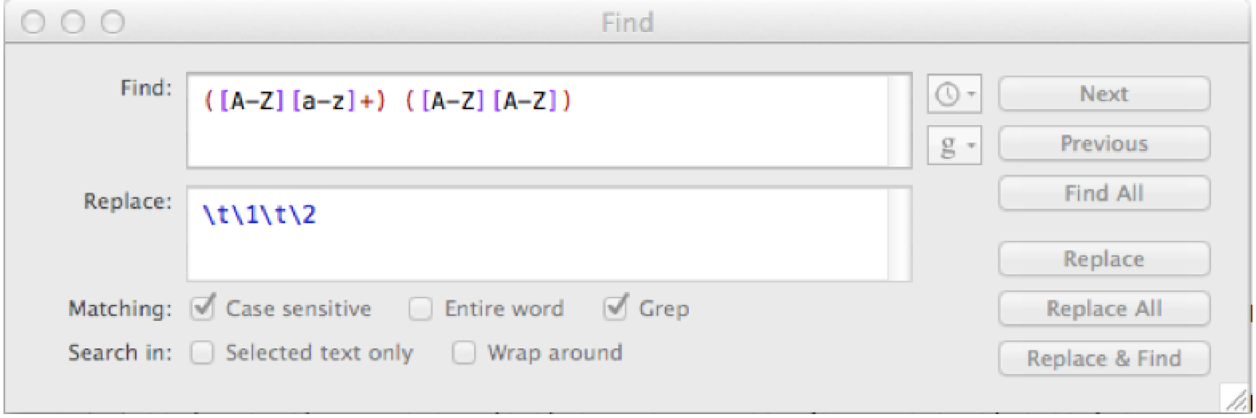
Matching: □ Case sensitive □ Entire word ☑ Grep

Search in: □ Selected text only □ Wrap around

18. Notice that the round brackets around the area code appear in blue, indicating that we have escaped them out. Click Replace All and the phone numbers will be set off with tabs.
19. Finally, let's put a tab before the name of the city. Normally, this would be tricky but in this case we know that the city name in our list is always followed by a two-upper-case-

character province abbreviation, in this example, ON. So the pattern we want to find is a capitalized city name, followed by a space, then two upper-case letters.

20. In the Find box, type the expression **[A-Z][a-z]+ [A-Z][A-Z]**. This will search for a single capital letter followed by any number of lowercase letters, then a space, then two capital letters. We'll use separate backreferences for the city and provincial acronym and put a tab between them when we recall them. Also, we need to tell TextWrangler that our phrase is case-sensitive so click that checkbox. The dialog should look like this:



Find: `([A-Z][a-z]+) ([A-Z][A-Z])`

Replace: `\t\1\t\2`

Matching: ☑ Case sensitive □ Entire word ☑ Grep

Search in: □ Selected text only □ Wrap around

21. Notice we didn't need another tab at the end of the replace phrase because we had already put in tabs before the postal codes. Click Replace All and the city name and province acronyms should be set off with tabs.

22. Hit COMMAND-A to highlight all the text and COMMAND-C to copy it. Then paste it into an empty Excel spreadsheet. We should now have a lovely, structured dataset, ready be pasted into Excel.

Tutorials and reference material for using Regular Expressions at regular-expressions.info.

More RegEx practice:
bitly.com/CUregex

Regular Expressions (RegEx) - Quick Reference

Match anywhere: By default, a regular expression matches a substring *anywhere* inside the string to be searched. For example, the regular expression **abc** matches abc123, 123abc, and 123abcxyz. To require the match to occur only at the beginning or end, use an anchor.

Escaped characters: Most characters like abc123 can be used literally inside a regular expression. However, the characters **\.*?+[{|()^$** must be preceded by a backslash to be seen as literal. For example, **\.** is a literal period and \\ is a literal backslash. Escaping can be avoided by using \Q...\E. For example: \QLiteral Text\E.

Case-sensitive: By default, regular expressions are case-sensitive. This can be changed via the "i" option. For example, the pattern **i)abc** searches for "abc" without regard to case. See options for other modifiers.

Commonly Used Symbols and Syntax

.	A dot matches *any single* character (except newline: `r and `n). For example, **ab.** matches abc and abz and ab_
*****	An asterisk matches zero or more of the preceding character, class, or subpattern. For example, **a*** matches ab and aaab. It also matches at the very beginning of any string that contains no "a" at all. **Wildcard:** The dot-star pattern **.*** is one of the most permissive because it matches zero or more occurrences of *any* character (except newline: `r and `n). For example, **abc.*123** matches abcAnything123 as well as abc123.
?	A question mark matches zero or one of the preceding character, class, or subpattern. Think of this as "the preceding item is optional". For example, **colou?r** matches both color and colour because the "u" is optional.
+	A plus sign matches one or more of the preceding character, class, or subpattern. For example **a+** matches ab and aaab. But unlike **a*** and **a?**, the pattern **a+** does not match at the beginning of strings that lack an "a" character.
{min,max}	Matches between *min* and *max* occurrences of the preceding character, class, or subpattern. For example, **a{1,2}** matches ab but only the first two a's in aaab. Also, {3} means exactly 3 occurrences, and {3,} means 3 or more occurrences. Note: The specified numbers must be less than 65536, and the first must be less than or equal to the second.
[...]	**Classes of characters:** The square brackets enclose a list or range of characters (or both). For example, **[abc]** means "any single character that is either a, b or c".

	Using a dash in between creates a range; for example, **[a-z]** means "any single character that is between lowercase a and z (inclusive)". Lists and ranges may be combined; for example **[a-zA-Z0-9_]** means "any single character that is alphanumeric or underscore". A character class may be followed by *, ?, +, or {min,max}. For example, **[0-9]+** matches one or more occurrence of any digit; thus it matches xyz123 but not abcxyz. The following POSIX named sets are also supported via the form [[:xxx:]], where xxx is one of the following words: alnum, alpha, ascii (0-127), blank (space or tab), cntrl (control character), digit (0-9), xdigit (hex digit), print, graph (print excluding space), punct, lower, upper, space (whitespace), word (same as \w). Within a character class, characters do not need to be escaped except when they have special meaning inside a class; e.g. **[\^a]**, **[a\-b]**, **[a\]]**, and **[\\a]**.
[^...]	Matches any single character that is **not** in the class. For example, **[^/]*** matches zero or more occurrences of any character that is *not* a forward-slash, such as http://. Similarly, **[^0-9xyz]** matches any single character that isn't a digit and isn't the letter x, y, or z.
\d	Matches any single digit (equivalent to the class **[0-9]**). Conversely, capital \D means "any *non*-digit". This and the other two below can also be used inside a class; for example, **[\d.-]** means "any single digit, period, or minus sign".
\s	Matches any single whitespace character, mainly space, tab, and newline (`r and `n). Conversely, capital \S means "any *non*-whitespace character".
\w	Matches any single "word" character, namely alphanumeric or underscore. This is equivalent to **[a-zA-Z0-9_]**. Conversely, capital \W means "any *non*-word character".
^ **$**	Circumflex (^) and dollar sign ($) are called *anchors* because they don't consume any characters; instead, they tie the pattern to the beginning or end of the string being searched. **^** may appear at the beginning of a pattern to require the match to occur at the very beginning of a line. For example, **^abc** matches abc123 but not 123abc. **$** may appear at the end of a pattern to require the match to occur at the very end of a line. For example, **abc$** matches 123abc but not abc123. The two anchors may be combined. For example, **^abc$** matches only abc (i.e. there must be no other characters before or after it). If the text being searched contains multiple lines, the anchors can be made to apply to each line rather than the text as a whole by means of the "m" option. For example, **m)^abc$** matches 123`r`nabc`r`n789. But without the "m" option, it wouldn't match.
\b	\b means "word boundary", which is like an anchor because it doesn't consume any characters. It requires the current character's status as a word character (\w) to be

	the opposite of the previous character's. It is typically used to avoid accidentally matching a word that appears inside some other word. For example, **\bcat\b** doesn't match catfish, but it matches cat regardless of what punctuation and whitespace surrounds it. Capital \B is the opposite: it requires that the current character *not* be at a word boundary.
|	The vertical bar separates two or more alternatives. A match occurs if *any* of the alternatives is satisfied. For example, **gray|grey** matches both gray and grey. Similarly, the pattern **gr(a|e)y** does the same thing with the help of the parentheses described below.
(…)	Items enclosed in parentheses are most commonly used to: • Determine the order of evaluation. For example, **(Sun|Mon|Tues|Wednes|Thurs|Fri|Satur)day** matches the name of any day. • Apply *****, **?**, **+**, or **{min,max}** to a *series* of characters rather than just one. For example, **(abc)+** matches one or more occurrences of the string "abc"; thus it matches abcabc123 but not ab123 or bc123. • Capture a subpattern such as the dot-star in **abc(.*)xyz**. For example, RegExMatch() stores the substring that matches each subpattern in its output array. Similarly, RegExReplace() allows the substring that matches each subpattern to be reinserted into the result via backreferences like $1. To use the parentheses without the side-effect of capturing a subpattern, specify **?:** as the first two characters inside the parentheses; for example: **(?:.*)** • Change options on-the-fly. For example, **(?im)** turns on the case-insensitive and multiline options for the remainder of the pattern (or subpattern if it occurs inside a subpattern). Conversely, **(?-im)** would turn them both off. All options are supported except DPS\`r\`n\`a.
\t **\r** **etc.**	These escape sequences stand for special characters. The most common ones are **\t** (tab), **\r** (carriage return), and **\n** (linefeed). In AutoHotkey, an accent (\`) may optionally be used in place of the backslash in these cases. Escape sequences in the form \xhh are also supported, in which *hh* is the hex code of any ANSI character between 00 and FF. In v1.0.46.06+, **\R** means "any single newline of any type", namely those listed at the \`a option (however, \R inside a character class is merely the letter "R"). In v1.0.47.05+, **\R** can be restricted to CR, LF, and CRLF by specifying (*BSR_ANYCRLF) in uppercase at the beginning of the pattern (after the options); e.g. **im)(*BSR_ANYCRLF)abc\Rxyz**

Greed: By default, *****, **?**, **+**, and **{min,max}** are greedy because they consume all characters up through the *last* possible one that still satisfies the entire pattern. To instead have them stop at the *first* possible character, follow them with a question mark. For example, the pattern **<.+>** (which lacks a question mark) means: "search for a <, followed by one or more of any character, followed by a >". To stop this pattern from matching the *entire* string <em>text</em>, append a question mark to the plus sign: **<.+?>**. This causes the match to stop at the first '>' and thus it matches only the first tag <em>.