Grep and Regular Expressions, Part 2

Grep and Regex, Part 2

Boundaries and Other Shortcuts

When using regular expressions, we'll often want to search for strings that are at the very beginning or end of a line, or are separated by word boundaries (to avoid getting "beard" when we search for "bear"). To search for these things, we'll use the boundary symbols `^` for the beginning of the line, and `$` for the end of the line.

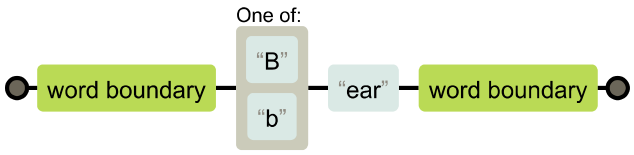
For example, if we wanted to search Grimm's fairy tales for all lines that begin with "the", we'll use the regular expression `^the`


```
grep --color=auto "^the" grimm.txt
```

- **the**re was nothing to do.
- **the**ir side, the stag leapt merrily by them, and the birds sat still upon
- **the** boughs, and sang whatever they knew.

Normally, if we wanted to search for a string at the end of a line, like "ing", we could use `ing$`. Unfortunately, due to a formatting quirk of books from Project Gutenberg, this won't work as normal!

The other very important boundary is the word boundary, which is represented with `\b`. It involves two characters, the slash and the "b", but gets interpreted as referring to just one search character: anything that signals the edge of a word. this will let us solve our "beard" problem.




```
egrep --color=auto "\b[Bb]ear\b" grimm.txt
```

Tabs, new lines, and "carriage return"

In addition to spaces, which we can include in a regular expression just by typing a space, the other kinds of important "whitespace" characters are tabs, new lines, and for some systems a "carriage return". The symbols in regex for these are

- `\t` : tab
- `\n` : new line
- `\r` : carriage return

"Carriage returns" are named after how typewriters work, and do basically the same thing as a new line. They're mostly only found in documents generated in Windows.

If you want to find two strings separated by whitespace, and you don't care if it was a tab, space, new line or carriage return, you can use the shortcut `\s`.

Other Shortcuts

The word boundary symbol `\b` and white space symbol `\s` are handy shortcuts. A few other often used shortcuts are:

- `\d` : Any number. Equivalent to `[0-9]`
- `\w` : Any alphanumeric character and underscore. Equivalent to [`a-zA-Z0-9_`].

Counting words of different lengths

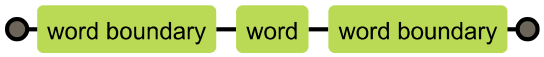
So, for example, let's say we wanted to see how common 1 letter words were compared to 2 letter words, and so on, in a text like Grimm's Fairy tales. So far we've been looking at the whole line being printed out when we search for a word like "bear":

```
egrep --color=auto "\bbear\b" grimm.txt
```

If we add the flag `-o` to the call to `egrep`, it will print out just the matching string, each one on its own line. With this, we can pipe the output of `egrep` to `wc`, and get word counts.

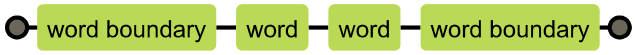
```
egrep -o --color=auto "\bbear\b" grimm.txt
```

So, let's find out how many single character words there are with `\b\w\b`

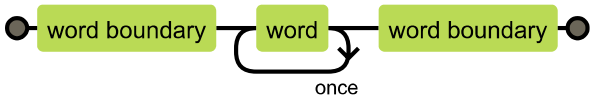



```
% egrep -o --color=auto "\b\w\b" grimm.txt | wc
    3693    3693    7386
```

We can search for two character words in one of two ways. First we could just enter `\w` twice, with `\b\w\w\b`



Or, we could tell the `\w` to loop twice, with `\b\w{2}\b`



Let's use this second option, and get the number of two character words.

```
% egrep -o --color=auto "\b\w{2}\b" grimm.txt | wc
   16555   16555   49665
```

Now three character words!

```
% egrep -o --color=auto "\b\w{3}\b" grimm.txt | wc
   31499   31499  125
```

If we keep doing this for Grimm's fairy tales, we get a pretty interesting distribution

word length	Number of words
1	3,693
2	16,666
3	31,488
4	24,501
5	11,418
6	7,416
7	5,069
8	2,672
9	1,454

10	673

Variables

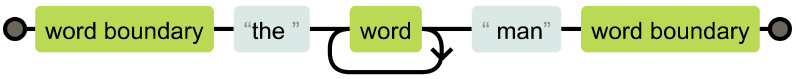
The last remaining important characters in Regular Expressions are `.`, `+`, `?`, and `*`.

Dot: `.`

The dot, `.`, stands for "any character". It could be a number, punctuation, a space, or anything. So, let's say you're playing Wordle, and you've figured out that the word ends in "ight", you could search a large book for inspiration for what the first letter is by using the regular expression `\b.ight\b`.

Plus `+`

The `+` operator means "one or more of the previous character". For example, let's say that we wanted to figure out what adjectives modify "man" in Grimm's fairy tales. We could look for all instances of "the ADJECTIVE man". We know we want there to be a word in between "the" and "man", but we don't know ahead of time how long we want it to be. We can use the regular expression `\bthe \w+ man\b` to get all of these phrases.




```
% egrep -o --color=auto "\bthe \w+ man\b" grimm.txt
the young man
the young man
the young man
the young man
the young man
the happiest man
```

Star `*`

The `*` operator means "find zero or more of the preceding character. So, if we wanted to know how often they mentioned "the (adjective) man", whether or not there was anything in between

"the" and "man". We could try using the regular expression `\bthe .*man\b`.


```
% egrep -o  --color=auto "\bthe .*?man\b" grimm.txt
the young man
the young man
the king was obliged to keep his word, and away went the young man
the princess, the horse, and the bird.’ ‘Ah!’ said the young man
the young man
the horseman
```

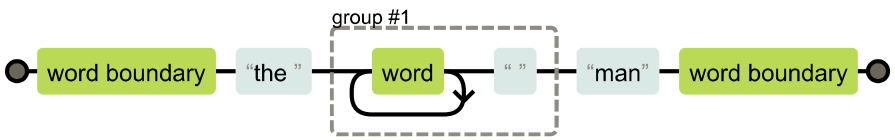
This hasn't really gone according to plan. Part of the problem is that `*` is "greedy", meaning it will try to find the longest possible string that matches. Second, we weren't precise enough with our regex and have examples like "horesman".

Question Mark `?`

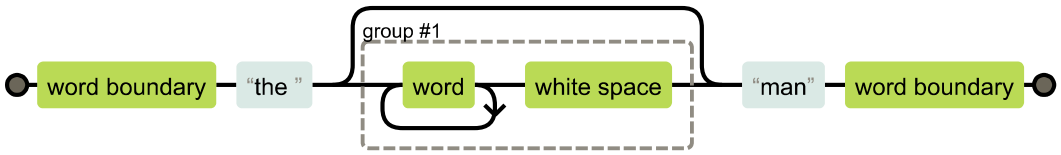
The `?` operator means "the preceding expression was optional. In the previous module, it was mentioned how `favou?r` would match both the American spelling "favor" and the British spelling "favour".

We can use `?` to solve our problem of looking for "the man" with an optional adjective in between. Let's look at our regular expression to find "the ADJECTIVE man" first, `\bthe \w+ man\b`

What we want to say is that the repeating word character, and its following space, are optional. First step is to "group" those parts of the expression together with parentheses: `\bthe (\w+ )man\b`



Then, we can make that group optional, by following it with `?`: `\bthe (\w+ )?man\b`



If we pass this to `egrep`, we get what we're looking for.

```
% egrep -o  --color=auto "\bthe (\w+\s)?\bman\b" grimm.txt
the young man
the young man
the young man
the young man
the young man
the happiest man
the man
```