

## Re: Teach me how to fish, regexp

**Source:** <http://coding.derkeiler.com/Archive/Perl/comp.lang.perl.misc/2003-10/0886.html>

---

**From:** Henry ([henryn\\_at\\_zzzspacebbs.com](mailto:henryn_at_zzzspacebbs.com))

**Date:** 10/08/03

Date: Wed, 08 Oct 2003 19:21:02 GMT

Martien:

Thanks for your response on this thread:

in article [slrnbo6sk1.pv1.mgju@verbruggen.comdyn.com.au](mailto:slrnbo6sk1.pv1.mgju@verbruggen.comdyn.com.au), Martien Verbruggen at [mgju@tradingpost.com.au](mailto:mgju@tradingpost.com.au) wrote on 10/7/03 7:12 PM:

<snip> <snip>

>

>>>> *Seems the best way to deal with this is to slurp, and use "split" with the appropriate regexp. Wrinkle: I need to retain the section numbers in the return strings.*

>>>>

>>> *I would probably set the input record separator (\$/, see perlvar) to "", which will treat two or more consecutive newlines as the record separator. Then each record starts with the number you're interested in.*

>

>> *Right, that's what I finally did, in effect. (I did something similar at the "split".) But this isn't very robust, I think: it depends on some typist somewhere \_always\_ following the rules.*

>>

>> *I think you are saying that slurp mode may not be the best choice.*

>>

>> *As far as your setting*

>>

>> *\$/ = "";*

>>

>> *This is not exactly intuitive from the point of view of a newcomer. Sorry, could you help me understand (or give me a blind rule of thumb) how what looks like setting a variable to an empty string implies "two or more successive newlines"?*

>

> *The perlvar documentation explains what \$/ (the input record separator) does, and that it has a "special" setting of the empty string, which makes it reads "paragraphs", i.e. blocks of text separated by two or more newlines.*

Right. You are too polite, so I'll say it: I need to RTFM.

That said, I have to say I find the man pages the last place I want to look in terms of convenience. I'll have to find a web based... Done!

Ah, much easier. Here's a relevant extract.

...Setting to "" will treat two or more consecutive empty lines as a single empty line. Setting to "\n\n" will blindly assume that the next input character belongs to the next paragraph, even if it's a newline. (Mnemonic: / delimits line boundaries when quoting poetry.)

Quoting `_poetry_`?

The choice of "" as a special is clearly a choice of convenience on the part the people doing the internals— and has no particular mnemonic or symbolic values. I'm glad that's clear.

This gives me a clue to the magnitude of my task of learning perl to do useful work.

```
<snip>
>>
>>> ( # start capture
>>
>> Capture? I guess you mean the mysterious "save the stuff you match"
>> mechanisms I've found in some perl references. The explanations
>> I've found are very short and not very useful. Also: I find it
>> hard to discriminate between parens used for operation grouping and
>> this use.
>
> Yes. Capturing parentheses "save" whatever is matched between them,
> and return it as a result of the operation, as well as in the named
> variables $1, $2, etc.. At the same time they group multiple
> characters together to form a single subpattern.
```

(I have the books "Perl by Example" and "Learning Perl" and what I've finding is those aren't particularly good references when it comes to details like this.)

Right, now I'm getting the idea. Parens capture the matched stuff.

Thanks for coming out and saying this so directly.

Apparently only specially-marked parens do NOT capture stuff, right. Also, I see by reading further that capturing is expensive in terms of processing time, so you might want to limit its use.

```
>
> There is more information about this in the perlre documentation,
```

Aha: in a Warning, I find

Re: Teach me how to fish, regexp

...(stuff)...

Captures stuff, but

...(?:stuff)...

doesn't. Cute.

> *well as in the perl documentation under the entry for*  
> *"m/PATTERN/cgimosx".*

Hmmm, yet another man page. How many are there? Over 100? Hmmm...

Yes, that's the best treatment of many issues that I could not understand by consulting other references. Thanks!

I'm still trying to figure out the best way of referring to what are here called "flags" in this context, but seem to have other names elsewhere: the "cgimosx" items. Am I confused, or is this confusing?

>

>>> *(?: # start grouping, but no capturing*

>>

>> *Sorry, could you speak more fully about this? Again, I haven't*

>> *found a good reference for this stuff.*

>

> *If you only want to group some stuff together in a subpattern, but you*

> *don't want that match of that subpattern returned as one of the digit*

> *variables, or in the return list, you use (?:PATTERN). Again, see the*

> *perlre documentation for a full explanation.*

>

>>> *.\| # literal . followed by two spaces*

>>

>> *Sorry, I don't get that. Could you explain more fully? I think that I*

>> *understand that a period, unescaped, matches any character, so I would*

>> *expect that you'd have to escape before the period to match a literal*

>> *period/decimal point.*

>

> *You're right. my mistake in transcribing the regular expression. there*

> *should be a backslash in front of the dot.*

Sound effect: <Long sigh of relief>. (I'm not a total dipstick.)

>

>>> *(.\*) # capture the rest of the record*

>>

>> *I think I understand that*

>>

>> *.\**

>>

>> *means "any character, repeated 0 or more times", but I don't get how the*

>> *parens lead to capture (and not operation grouping, as above) and eventual*

>> *appearance of the captured data somewhere.*

>  
> *It does both. They group, and as a side effect, the matched subpattern*  
> *gets captured and returned (in this case as the second element of the*  
> *returned list, as well as in \$2).*

Aha!

It seems to me that you can use parens to affect operation grouping, if you are sufficiently qualified (or ambitious); that's story #2. Depending on which documentation you use, you may discover Story #2, which describes how parens `_also_` store data, how to access that data, and how and why avoid unnecessary use of this feature.

What I did not find —part of why I'm here— is a reference that tells both stories.

>  
>>> *The first capturing set of parentheses returns the paragraph*  
>>> *number, including the sub-number, if present, and the second*  
>>> *capturing parentheses set returns the "Blah, blah.." bit up to the*  
>>> *end of the record.*  
>>  
>> *Right, as I said above, I can't figure out how this aspect works.*  
>> *This may seem obvious to you but looks like a hidden (or magical)*  
>> *side-effect to me.*  
>  
> *The fact that those grouped subpattern matches get returned (and saved*  
> *in \$1, \$2...) is more an effect of the m// operator (documented in*  
> *perlop) than of regular expressions themselves. However, they do get*  
> *captured in regular expressions, and you can refer back to them (with*  
> *\1, \2...) inside of the same regular expression.*

I'm sorry you said `_that_`, because now I have uncertainty about the scope of this "side effect" in different contexts. I guess your purpose is to alert me to the fact that parens in regexps anywhere do save data, but the accessibility of the data varies from context to context. Right?

\  
>  
<snip>  
<snip>  
>  
<snip>

> *There are also a perlrequick and a perlretut manual page, which are*  
> *more gentle introductions to regular expressions than the perlre*  
> *reference documentation. You should probably have a bit of a read of*  
> *those.*

>  
In my spare time, I'll concatenate all the perlxxxx man pages and see how the contents compare to a moderate sized book.

That's a lot of stuff, and in a reference format. Fortunately, the examples are generally quite good, but this isn't exactly the most friendly environment.

>

> *Furthermore: Don't worry too much that some of this stuff looks*

> *magical. It is. Perl is full of things that you just have to learn*

> *about by immersion, and by repeated visits to the same documentation.*

> *it can take a while before some of this stuff becomes automatic.*

Perl doesn't seem to be anything one can pick up quickly, that's for sure.

Thanks,

Henry

henryn@zzzspacebbs.com remove 'zzz'

>

>

> *Martien*