

Named Captures in 5.9.5

Forget about \$1 and \$2, here's \$+{my_name}

by brian d foy
brian.d.foy@gmail.com



Perl 5.9.5, the latest development release of Perl and the one that will probably turn into Perl 5.10, has some very exciting new regular expression features.

■ The old way -----

Up to now, when I use memory parentheses, I have to remember which of the number variables (\$1, \$2, ...) the matches will be in, then use those appropriately in my code:

```
$_ = 'Just another Perl hacker,';

if( /Just another (\w+) hacker,/ ) {
    print "I found a $1 hacker!\n";
}
```

I could fix that up to immediately assign the number variables to variables with better names so I only have one place to update them:

```
$_ = 'Just another Perl hacker,';

if( /Just another (\w+) hacker,/ ) {
    my( $type ) = ( $1 );
    print "I found a $type hacker!\n";
}
```

Either way, when I change the regular expression to include more capture parentheses, I have to remember to update the code for the new position:

```
$_ = 'Just another Perl hacker,';

if( /Just (\w+) (\w+) (\w+),/ ) {
    print "I found a $2 hacker!\n";
}

if( /Just (\w+) (\w+) (\w+),/ ) {
    my( undef, $type, $str ) = ( $1, $2, $3 );
    print "I found a $type hacker!\n";
}
```

■ Named captures -----

Number variables are old school, though. For years I've wanted a way to give names to captures so it's all in the regex and I don't depend on the order of parentheses. With Perl 5.9.5, I can

use the (?<NAME>PATTERN) sequence. I put the name I want to use inside literal angle brackets, and if the match succeeds, the new special hash %+ hash has a key for that name:

```
use Data::Dumper;
$_ = 'Just another Perl595 hacker,';

if( /(?<word>\w+) (?<type>\w+) (\w+),/ ) {
    print "I found a $+{type} hacker!\n";
}

print Dumper( \%+ );
```

The output shows that %+ has keys for word and type:

```
I found a Perl595 hacker!
$VAR1 = {
    'word' => 'another',
    'type' => 'Perl595'
};
```

The new %+ variable acts just like the other match variables: Perl clears it and sets its value only on a successful match. The hash, however, will only have keys for the defined captures.

```
use Data::Dumper;
$_ = 'Just another Perl hacker,';

if( /(?<word>\w+) (?<type>\w+) (\w+), (?<after>.+)?/ ) {
    print "I found a $+{type} hacker!\n";
}

print Dumper( \%+ );
```

Since I use the (?<after>.+)? after the literal comma, I have a possible capturing group. The ? at the end makes it optional, so if this group doesn't match anything, I won't see a key in %+ and the output of the program is the same:

```
I found a Perl hacker!
$VAR1 = {
    'word' => 'another',
    'type' => 'Perl595'
};
```