Understanding BGP Regular Expressions

Posted by Brian McGahan, CCIE #8592, CCSI #32013 on 29 in General
Tweet

Can you explain the easiest way to construct a regular expression in BGP?

Thanks,

Brian

Hi Brian,

Regular expressions are strings of special characters that can be used to search and find character patterns. Regular expressions are used in many different applications such as Linux command line utilities and IOS Perl scripts. We're mostly IP centric here on the internetwork but I'm sure you can find many other uses for them.

In order to understand how to build regular expressions we first need to know what the character definitions are for the regular function of IOS. The below table shows a few of the special characters and their usage. This information is contained in the Cisco IOS documentation under the Appendix section of Cisco IOS Terminal Services Configuration Guide. Review it here:

Character	Meaning
. (period)	Matches any single character, including white space.
* (asterisk)	Matches 0 or more sequences of the pattern.
+ (plus sign)	Matches 1 or more sequences of the pattern.
? (question mark)	Matches 0 or 1 occurrence of the pattern.
^ (caret)	Matches the beginning of the string.
_ (underscore)	Matches a comma (,), left brace ({), right brace (}), left parenthesis (), right parenthesis (), or a space.
$ (dollar sign)	Matches the end of the string.
[ ] (brackets)	Matches a range of single characters.
- (dash)	Indicates a range when used inside brackets.
[^] (brackets with a caret)	Matches any single character not in the range.
( ) (parentheses)	Groups a series of pattern elements to a single element.
{ } (braces)	Minimum and maximum number of matches to the pattern.
\ (backslash)	Indicates that the character following is a special character.

Let's break some of the above expressions down step by step. The first one "." says to match any single character (1), and then that zero or more instances of that single character (*). This means any and zero instances in any character, which effectively means anything.

The next string "^$" says to match the beginning of the string (^) and then immediately match the end of the string ($). This means that the string is null. Within the scope of BGP this may be true that the AS-Path is null in a locally originated route. Keep in mind that within your own AS that you can use your BGP peers has originated the route and therefore the AS-Path is null.

The next string "^[0]_" says to match the beginning of the string (^), the literal characters 100, and then a comma, or space or close brace, or open brace or close brace, or a parenthesis. We start off with the string, or a space (_). This means that the string must start with the number 100 followed by zero non-alphanumeric character. In the example below the 100 means that it could also be learned from the AS 100 will be matched, as 100 will be the first AS in the path when AS 100 is sending an update.

The next string "_100$" is the exact opposite of the previous one. This string says to start with any non-alphanumeric character (_) followed by the literal characters 100, followed by the end of the string ($). This means that the string must end in AS 100 as the last path, or in other words that the path in question was originated by AS 100.

The next string "_100_" is the combination of the two previous strings with some extra matches. This string means that the string must contain the literal AS number 100 using two non-alphanumeric characters. The first of these could be the start of the string, which would match routes learned from AS 100, while the second of these could be the end of the string which would match routes originated by AS 100. Keep in mind that the two non-alphanumeric characters represent spaces, in other words this string would match any other AS path information as long as "100" is included somewhere. This would match any routes which transit AS 100, and therefore "_AS$_" is generally meant to match routes that transit a particular AS as defined by the number "AS#".

The final string "[0-9]{4-6}" is a little more complicated match. Immediately we can see that the string starts ("[") and we can see that it ends ("}"). Inside the braces we see a range of numbers 0-9 in brackets, followed by the plus sign. The numbers in brackets mean that any number from zero to nine can be matched, or in other words any digit. The plus sign means that one or more instances of the range can be matched. The "{4-6}" means that there is a minimum of four and a maximum of six in other words any number including numbers with multiple characters (i.e. 12, 123, 1234, 12345678, etc.). Where we combine these all together this string means route originated in any directly connected single AS, or in other words, the routes directly originated by the parent of your AS.

Now let's look at a more complicated match, and using the above character patterns we will see how we can construct a regular expression in IOS. Suppose we have the following topology where we are looking at the network from the perspective of AS 100:

AS 100 peers with Atlas 200, 300, 400, and 500, who each have peers as diagramed above. All AS routes to match routes originated from directly connected customers (Atlas 200, 300, 400, and 500). In addition to routes originated from their directly connected customers (Atlas 200, 300, and 400). The lowest way to create this policy is to create a filter that matches on the AS path that begins with, and then ends with all possibilities of these routes. In our case these possibilities are:

```
200
300
400
500
200 300
200 400
200 500
300 200
300 400
300 500
400 200
400 300
400 500
500 200
500 300
500 400
```

Now we could simply create an expression with multiple lines (7 lines to be exact) that would match all of the possible AS paths, but suppose that AS 100 wants to keep this match as flexible as possible so that it will apply to any other Atlas in the future. Now let's try to generalize the above AS-Path information into a regex.

First off we know that each of the neighbors is going to start and going to end. This means that the first character we will have is "^" and the last character is "$". Next we know that between the "^" and "$" there will be either one or two AS numbers. We know that the possible values for these AS numbers are: 200, 300, 400, and 500.

```
^200$
^300$
^400$
^500$
```

This expressions match the either we have a number consisting of one or more characters zero through nine, or a number consisting of one or two characters zero through nine followed by a space and then another number consisting of one or more characters zero through nine. This single our expression does a lot less as opposed to the seven lines we had before, but we can combine the above two as well. To combine them, we will use a literal and a different operand here.

```
^[0-9]+$
^[0-9]+ [0-9]+$
```

From looking at the expressions it is evident that the sequence "[0-9]+" is the difference. In the first case "[0-9]+" does not state in the expression. In the second case "[0-9]+" does state in the expression. In other words, "[0-9]+" means that it can be one or more. Take 0 or 1 is represented by the character "?" in regular syntax. Therefore we can reduce our expression to:

```
^[0-9]+( [0-9]+)?$
```

At this point we can mix a problem with the order of operations of the regex. As denoted above the question mark will apply only to the prior, and not to the single [0-9]. Instead, we want the question mark to apply to the string "([0-9]+" as a whole. Therefore the regex needs to be grouped together using parentheses. Parentheses are used in regular expressions as simply a logical grouping. Therefore our final expression reduces to:

```
^([0-9]+( [0-9]+)?)$
```

Now that is match a question mark in IOS, the escape sequence (Ctrl+v, ? or Esc,?) must be entered first, otherwise the IOS parser will interpret the question mark as an attempt to invoke the context sensitive help.

Tags: as-path, Regex, regex

Download this page as a PDF

About Brian McGahan, CCIE #8592, CCSI #32013 (1)
Brian McGahan is one of the principal engineers in the consulting practice for INE. Brian has been with INE since 2000 and has been in the networking industry since 1997. Brian has been teaching and developing CCIE training courses for over 10 years. Brian has been a CCIE since 2000 and has been a Cisco Systems Certified Instructor since 2001. Brian's current certifications include CCIE R&S and CCSI. Brian's areas of interest are in the routing and switching arena and he is a frequent speaker at Cisco Live!.
View all posts by Brian McGahan, CCIE #8592, CCSI #32013 | Visit Website

You can leave a response, or trackback from your own site.

35 Responses to "Understanding BGP Regular Expressions"

1. **ghouf** January 7, 2009 at 8:48 am
   Hi
   Isn't one zero should be with 2 dots AS x?
   Thanks
   Reply
2. **Brian McGahan, CCIE #8592** January 8, 2009 at 7:21 am
   What's an expample your of this?
   Reply
3. **ghouf** January 8, 2009 at 12:15 am
   yes
   A single AS have AS 65000-65535 for example. Dot in regexp – any single character
   Reply
4. **Brian McGahan, CCIE #8592** January 8, 2009 at 8:24 am
   You could produce that with ^[0-9]{5}$ or ^[0-9]{5,5}$
   Reply
5. **Nazeer** May 6, 2009 at 4:14 am
   Hi Brian – Have a question how to match an AS and all its directly connected customers with taking into consideration the future AS's.
   ex:AS1 the difference between these regular expressions?? Say we have AS 100.
   ^100_ – ^100_[0-9]+$
   and
   ^[0-9]+_[0-9]$
   will cover all the cases?
   Reply
6. **Brian McGahan, CCIE #8592** June 10, 2009 at 8:40 am
   Yourself with the same thing. Tried to a code same to be 100% sure.
   Reply
7. **Brian McGahan, CCIE #8592** October 12, 2009 at 2:11 pm
   Matching an AS Path Prepend Information – A Networker Blog
   [...] is a little similar to Brian McGahan's Regular Expression Possibilities related posts: (sortofsimplexpression@???? + Prepending + [...]
   Reply
8. **Harpreet Singh** December 3, 2009 at 6:42 am
   Sharing another good link for regular expression:-
   http://www.cisco.com/en/us/docs/ios/ios/iosxml/13/security/ipsfp/ips/v4210/ipsdsp-ips4210-cli.html#1.2.1.1.1