DMA XXX: Regex for Discord Automod 101

By @naviking

CW: This document may include objectionable content such as slurs for educational purposes

Discord's built-in automoderator is a powerful tool for blocking unwanted messages on your server. While there is already an excellent article discussing the basics of automoderator, the feature that makes it so powerful is also the feature that is the most complicated to use: regex support.

This article will discuss what regex is and the basics of using it with Discord's automoderator, along with some example regexes and additional resources for learning regex.

What is Regex?

Regex is actually shorthand for regular expressions. Compared to a discrete banned words list, regex instead attempts to match **patterns** of characters in a message, providing greater flexibility for matching text strings. Regex accomplishes this by interpreting certain characters in your regex text as modifiers to the characters around it. For example, the + sign is used to indicate to the regex parser to match that character if it is repeated one or more times. We will discuss these special syntaxes later.

Regex is supported in nearly all major programming languages such as Javascript, Python, and more. However, not all kinds of regex are created equal. While many regex characteristics are interchangeable between programming languages, you'll find that some kinds of regex support syntax that others don't or indicate certain patterns with different syntax compared to others.

For the purposes of this article, we will be discussing the Rust kind of regex because this is what Discord uses for automoderator.

Regex Syntax

First you must understand how regex interprets various characters and text strings. This is not a complete list of all supported special characters but contains the ones that are the most useful. You can see additional examples and explanations here.

Special Syntax	Description	Example Regex	Example Matches
\w	Matches any word character, consisting of a-z, 0-9, and an underscore. **Note that discord compiles this by expanding it out to every word character there is. This can result in an error message that your regex is too long, even when it's not**. You can approximate \w with [0-9a-z_] (explained a bit later). Case sensitivity is not necessary when applying this to Discord's automoderator.	\w	a 3 _
\W	Matches any non-word character. **This has a similar issue that \w does.** You can approximate it with [^0-9a-z_] .	\W	- &
\b	Matches a word boundary, consisting of a word character on one side and a non-word character on the other. Note that an underscore is considered a word character. **This is a foundational component of making sure your regexes work as expected. While Discord's banned words are automatically interpreted as exact matches, regex is automatically interpreted as "containing the string anywhere in the word."** **To put it simply entering "banana" as a banned word would not match a message of "I ate some bananas" in it because bananas =/= banana, but entering "banana" as regex would match such a message.** **Failing to use word boundaries in your regex can result in regex that are much more aggressive than expected.** For clarity the subsequent example matches will be written to match the entire string, but without word boundaries in the example regex you could put any kind and amount of text before or after the example and it would still technically match.	\babc	abc abcd (note: only the abc portion is matched, but the regex parser will still return that a match was found and act accordingly) **Does not match** 1abc

Special Syntax	Description	Example Regex	Example Matches
+	Matches one or more of the character before it	a+	aaaa aa
?	Matches zero or one of the character before it	ab?	a ab
*	Matches zero or more of the character before it	ab*	a abbb
{X,Y}	Matches a number of repetitions of the character before it between X and Y. You can omit either X or Y to match "up to Y" repetitions or "at least X" repetitions, but you must always include the comma. **Discord's automoderator limits the maximum repetitions to 168**.	a{5,}	aaaaaaa aaaaa
|	Matches the pattern on either side, essentially an "or" declaration	abc|def	abc def
()	Enclosing regex in parentheses indicates that the regex parser should resolve the expression in the parentheses to determine a match. This is referred to as a capture group. Repetition-type syntax such as *, +, ? and {} apply to the whole capture group. Nested parentheses and brackets are supported, however, **Discord's automoderator does not support more than 10 levels of nested capture groups/brackets**	(hugg|kiss)(ing|ed)+	hugging hugged kissing kissed kissinging

Special Syntax	Description	Example Regex	Example Matches
[]	Enclosing regex in brackets indicates that the regex parser should match any character in the brackets. **Don't forget that the parser is interpreting individual characters, not words**.	[cat]	c a t
[X-Y]	Including a starting character (X) and ending character (Y) separated by a hyphen within brackets indicates that the regex parser should match any characters between those characters (inclusive). Your starting character must come before your ending character according to Unicode, and you can also define a character with its unicode value by using \u{value} . For example, \u{0021} would match an exclamation mark. Remember, text in brackets is an "or" type match by character, so you can also include multiple ranges within brackets. Repetition-type syntax such as *, +, ? and {} apply to the whole capture group as well.	[0-9a-z]	1 2 3 d y
[^X]	Similar to [] but matches characters that are **not any** of the characters in brackets. This also works with ranges.	[^0-9]	a -
\n	Represents a newline character, essentially an invisible character that is inserted when you make a new paragraph by pressing Enter (or in the Discord message box, Shift + Enter).	a\nb	a b
\s	Represents any whitespace character such as a space, newline, tab, etc.	a\sb	a b
\S	Represents any non-whitespace character (the opposite of \s)	a\Sb	adb a-b

Special Syntax	Description	Example Regex	Example Matches
. (a period)	Matches any non-newline character (letters, numbers, punctuation, whitespace, etc.), essentially the opposite of \n. If you want this to additionally match newlines, use the Rust flag (?s) , including the parentheses, at the beginning of your regex.	a.b	adb --- a-b --- a b
\d	Represents a decimal digit (numbers 0 through 9).	\d	1
\D	Represents a non-decimal digit (the opposite of \d)	\D	w
^	Represents the start of a line. When using the (?s) flag, this only matches the start of the message rather than the start of each new line	^!	!Text --- **Does not match** Text!
$	Represents the end of a line. When using the (?s) flag, this only matches the end of the message rather than the start of each new line	!$	Text! --- **Does not match** !Text
\	The backslash by itself is an **escape character**. This is used when you want to **literally match a character or phrase that is otherwise special syntax**. If your special syntax has a backslash in it, you will need to include an extra backslash to escape it.	\\b	\b

Regex also uses a function called **flags** to define behavior for how a specific regex should be interpreted. Most importantly, **Discord enables the case insensitive flag (?i) by default**. This means you do not have to worry about matching messages in a case sensitive way (conversely, if you are using a standard regex tester such as Rustexp (lpil.uk) you will either need to enable this flag for your regex or ensure your testing text is all lowercase).

However, if you do want your regex to be case sensitive, **you can disable the case insensitive flag by beginning your regex with (?-i)**. If you decide to use Unicode ranges in your regex as described in the [] syntax, it's recommended to disable the case insensitive flag.

Regex Examples

With the explanation of regex syntax out of the way, it's time to discuss the most important part of regex: actually writing one! While some regex examples have been provided for purposes of explaining special characters, this section will walk you through creating a regex to match an actual word.

Let's start with a common homophobic slur: faggot

1. **Add word boundaries**: We might remove these later, but just to make sure we understand the scope of our regex, lets start by adding word boundaries on either site
   \bfaggot\b
2. **Consider repetition evasions**: In this case, modifying the number of g's or t's is a pretty obvious way to change how the word is spelled while not losing the meaning. Let's match the word if someone uses one or more g's.
   \bfag+ot+\b
3. **Consider omission evasions**: Oftentimes, removing vowels is a way to get around a word filter. Let's further modify the regex so that it will still match if 0 or more of any of the vowels are used.
   \bfa*g+o*t+\b
4. **Symbolic substitutions**: Substituting letters with numbers or symbols is a common way to evade word filters. Fortunately, we can use regex to match any set of characters we want by enclosing the set in brackets!
   \bf[a@4]*[g8]+[o0]*[t7]+\b
5. **Plurals and suffixes**: It's not too hard to simply make words plural. Since we want to keep word boundaries where possible, let's explicitly match the plural form of the word and versions with -ing or -ed at the end (in case someone thinks they're clever making it into a verb). Don't forget to consider the previous steps when adding plurals and suffixes! In this case, we'll add an optional capture group at the end (represented with the expression in parentheses followed by a question mark to match 0 or 1 of what's in the parentheses), and define each of the three things mentioned separated by |.
   \bf[a@4]*[g8]+[o0]*[t7]+([s5]|[1i]*n+[g]|[e3]*d)?\b
6. **Compound words**: In some cases, you may know that the word you're trying to match is creatively added to the beginning or end of a word. If you don't know the exact words it might be combined with, you could consider removing one or both word boundaries. However, if your word is very short, you must be careful about this or the chance for false positives becomes very high. In this case "faggot" is often used as a derogatory suffix to many words, so we will remove the beginning word boundary.
   f[a@4]*[g8]+[o0]*[t7]([s5]|[1i]*n+[g]|[e3]*d)?\b
7. **Further refinement**: This covers most of the basics and should be a good balance between true positives (matching a message when it contains your targeted word or phrase) and false negatives (failing to match the message when it contains your targeted word or phrase). However, if you are willing to risk more false positives (matching a

message when it didn't contain your targeted word or phrase) you can consider making your regex even more sensitive by adding more repetition operators to other letters in the phrase or further slicing it down. For example "fag"[1] is just as derogatory, so let's match that too by making the "ot" part optional as well. Let's do all of that here

f+[a@4]*[g8]+([o0]*[t7]+)?([s5]+|[1i]*n+[g]|[e3]*d)?\b

You can test your regex at Rustexp by pasting your regex into the regex box and typing your test text in the subject box. Wow, that's a lot of matches already!

Matching one or more words

When it comes to spam messages, you may find that the exact messages vary but that the content is still similar. In this case you might want to detect if multiple words or phrases are present in a message and act on that.

For this example, let's work on detecting **crypto scams**. Our archetypical message is something like the following

> I'll help 20 people on how to earn 1 ETH or more within 72hours from the Crypto Market, but you'll pay me 10% of your profit when you receive it. DM me to know more!

---

[1] That being the case, fag is short enough that it can randomly appear in things like web page URLs. Matching 3 or 4 letter words without word boundaries is often subject to false positive so you might want to do something like (\bfag|fag\b) so that it has to be at the beginning or end of a word and not in the middle..

First we need to start by understanding **the key components of this message** in order to properly match it. In this case, we can break it down to three components

1. It is about "earning" money or "helping" people
2. The amount and type of money earned is specified, and "crypto" plays a large part in this message
3. There is a call to action to contact the user

We know our components, but we don't know what will be between them. **This is when the \b.*\b syntax is useful**. This essentially matches "any number of non-newline characters between words." It is an effective way to essentially "ignore" the parts of the message between what we're specifically looking for so that if the components we define are listed in the order we watch them, regardless of what else is in the message we will still match it.

Knowing that, let's start building our regex

1. **(help|teach|earn|learn)** : This will match people saying they will "help you" or "teach you" or say that you will "earn" money.
2. **([^a-z^0-9]*[0-9]+k|crypto|bitcoin|eth|cash|cryptocurrency)** : This will match them describing any specific amount of money being earned (e.g. 30k, $30, or 10) or the type of money being earned (bitcoin, eth, cash, cryptocurrency) as well as an errant mention of crypto in the middle of the message.
3. **(dm|message|contact|write to|send|call)** : This matches various words that invite you to contact the user in question such as "message me," "write to me," "call me," etc.

Now let's put these together with the syntax above to get something like this

```
\b(help|teach|earn|learn)\b.*\b([^a-z^0-9]*[0-9]+k|crypto|bitcoin|eth|cash|cryptocurrency)\b.*\b(dm|message|contact|write to|send|call)\b
```

Finally, you may encounter cases where people will use ordered lists or paragraphs for messages like this. Because the period character matches non-newline character, the inclusions of newlines in the offending message may cause your regex not to match. Here you can use the (?s) flag to tell the regex engine to have the period character match newlines as well.

```
(?s)\b(help|teach|earn|learn)\b.*\b([^a-z^0-9]*[0-9]+k|crypto|bitcoin|eth|cash|cryptocurrency)\b.*\b(dm|message|contact|write to|send|call)\b
```

Excellent! You can use this as is or refine it further using the tricks from the previous example. **Do keep in mind that Discord limits the length of regex to 256-260 characters**, so you can't get too crazy with it.

Further Reading

This should give you everything you need to start writing regex on your own for Discord! However, if you want to start implementing more advanced concepts, I'd recommend reviewing

some of the following resources. Regex has more applications than just in Discord automod, so if you learn it now, you will thank yourself later for taking the time to understand it!

- Rust regex tester (mentioned previously): Rustexp (lpil.uk)
- Rust's Regex documentation: regex - Rust (docs.rs)
- Comprehensive general regex tutorial: Regex Tutorial—From Regex 101 to Advanced Regex (rexegg.com)
- Regex tester with explanations (just make sure you set it to Rust if you want to use it with automod): https://regex101.com/
- Automoderator configuration template by me (naviking) with regex and banned words examples: Automod Config - Template - Google Sheets

---

This document is a Discord Moderator Academy style article. With the closure of the various Discord Moderator Programs, it is unlikely that this will ever have the opportunity to be added to the Discord Moderator Academy. This article is presented as-is courtesy of @naviking. If you want to learn more about server moderation and meet other people who are contributing to thriving Discord communities, check out the Discord Networking Discord https://discord.gg/8QEdtxqTJQ or learn more about NaviKing on his website https://www.discomm.cc/