Automation for Beginners

Course Short Description

This course focuses on learning how to lessen the moderation burden through automation with a strong focus on learning how to use AutoModerator.

Course Long Description

Welcome to our course on automation for beginners. This course is intended for moderators who are looking to begin learning more about AutoModerator and bots after completing our Mod Certification 201 course. Automation can get complicated and there is a bit of a learning curve. That said, if you're willing to try new things and review documentation as needed, you can learn how to automate tasks in your community.

If you're already a mod bot and AutoMod wizard, you're not the intended audience for this course. However, you're still more than welcome to try it out, earn a trophy, and share feedback on what additions you think would be most helpful for those new to automation, or what you'd like your future mod recruits to learn about automation, at the end of the course.

By the end of this course you should:

- Understand what automation on Reddit is and when using it is appropriate
- Understand what AutoMod can and can not do
- Understand the different parts of an AutoMod rule and what they can do
- Be able to write a simple AutoMod rule from scratch
- Know some best practices around AutoMod rule crafting
- Know what Regex is and how it relates to AutoMod
- Have a basic understanding of how RegEx works in a simple keyword variation search
- Understand what mod bots are
- Know where to access resources around mod bots
- Understand what script files are and how they relate to mod bots

In this course, we also share some commonly used AutoMod rules, a couple of ways you may be able to accomplish common automation tasks with simpler native Reddit tools, and a bunch of AutoMod tips.

Intro to Automation for Beginners

Automation Basics

Automation tools can surface content for review, remove spam or highly reported items, and handle countless other common tasks. Learning about how to use automation tools and applying what you've learned can help reduce the amount of repetitive tasks and manual oversight required to run your community, which can have a strong positive impact on both your community and your moderation team.

When we say automation in this course, we're speaking of utilizing bots to reduce what human moderators need to do by having the bots carry out repetitive tasks that don't necessarily require human intervention. This gives your team more time to spend on community building and engagement which can be especially valuable when your community is very active.

Before we get started, note that any exercise or practice you do for this course should be done on the desktop site. You will not be able to effectively complete an exercise on a mobile device, at least not in a way that won't be a bit of a hassle. You should also already have AutoModerator set up in a Reddit **test community**, as was covered in Mod Certification 201. If you haven't, that's ok. You will be prompted to create a test community and set up AutoMod later in this course.

At the end of the course, we'll share a list of resources you can reference as needed. We'll also share relevant resources throughout the course. Please note that some of the resources provided are maintained by volunteer moderators or were recommended by volunteer moderators and are third party sites. Reddit is not responsible for their content.

Now we'll go over some basic information on bots.

Bot Basics

What are bots?

Bots are, at a very basic level, a set of instructions that are written in a programming language, and executed by a program.

You've probably heard the word 'bot' commonly referenced when people are talking about fake accounts on various social media platforms. In this course, we're not talking about bots created to mimic human accounts. We're focused on good bots! Bots that are programmed to manage moderation tasks that don't require direct human action for those tasks to be performed in an appropriate way.

Why use bots?

- Bots don't need sleep, or to go to work; they can function 24/7
- Bots are able to act quickly
- Bots don't get tired or bored and can complete repetitive tasks so you don't have to
- Bots can search for keywords more efficiently

When Not to Bot

While there are many instances when a bot can be useful in handling certain tasks, it's important not to overuse automation and recognize which tasks are best handled by a human moderator, or when bots and mods can work together, such as when bots are used to filter content for human review.

Communities need to feel like the community is cared for by people. A member will feel more supported when getting human responses when they modmail, for example. Too many automated replies won't be read and can feel like spam, even when set up by the moderators.

One of the most commonly used bots you'll see on Reddit is one available to all communities and one you should have at least some basic familiarity with if you're taking this course, and that's AutoModerator.

Bot Automation Alternatives

Before we dive into AutoModerator, we just want to touch on a couple of other native Reddit tools we covered in Mod Certification 201. If you completed that course, you are aware that tools such as 'Content Controls' and 'Crowd Control' make it easy to set up some tasks AutoModerator can do, especially if you aren't familiar with AutoMod.

For example, 'Content Controls' allow you to ban certain words, and 'Crowd Control' allows filtering of content from Redditors with negative karma. You can even use content controls in concert with AutoMod. And while these tools are easier to set up, especially for new moderators, AutoMod should be your go-to tool for more advanced or granular automated rules.

So, let's get to it!

AutoModerator

Getting Started With AutoMod

What is AutoModerator?

u/AutoModerator is Reddit's native moderation bot. It's available in every community and can be used by moderators to complete a variety of moderation tasks. The instructions (rules) for AutoMod are written in YAML and entered into a configuration wiki page within your community's mod tools. If you completed Mod Certification 201, you should at least have experience setting AutoModerator up and adding a rule.

Using AutoMod in a test community

Although not required, if you haven't already, we highly recommend that you at least go through the lesson and exercise for AutoModerator in Mod Certification 201 before moving forward, as you'll need to have AutoMod set up in a **test community** for this course. If you prefer, you can instead learn how to set up AutoMod by following the instructions in the Mod Help Center article.

If you don't have a test community yet, you should create one (commonly, people create test communities that mimic their username) or ask your moderation team if they already have one you can use. It's best to start with a clean AutoMod config wiki page when you're just learning how to write rules.

We strongly suggest **not** testing any of the rule examples or exercises from this course in an active community. If you still decide to test AutoMod in an existing community, you must get sign off from any co-mods before making any changes, as failing to do so could negatively impact your community and moderation team.

Review resources as often as needed

As you're learning about AutoMod, we recommend that you review the links provided within each lesson for further context on what types of things AutoMod can do. We often link out to AutoMod's full documentation in this course. Also be sure to check out the r/AutoModerator community. They are an unofficial Reddit community that provides excellent support for moderators with AutoMod related questions and they have helpful resources for mods in their wiki. You can also find a list of further AutoModerator resources in r/modguide.

Understanding Automod's Capabilities

What can AutoModerator do?

With the right set-up, it can:

- send a modmail to alert you to potential issues
- reply to posts with helpful comments, like pointing users to your community rules or wiki
- sticky comments
- remove or flair posts by domain, author, or keyword
- filter specific content for review
- require all comments to include the word "pancake"
- and take many, many other actions automatically

What can't AutoModerator do?

- see or flag reposts
- act on old or past content; AutoModerator can only act on new posts, comments, reports, or edits, and cannot delay an action
- react to vote counts, changes in flair, or a redditor's history

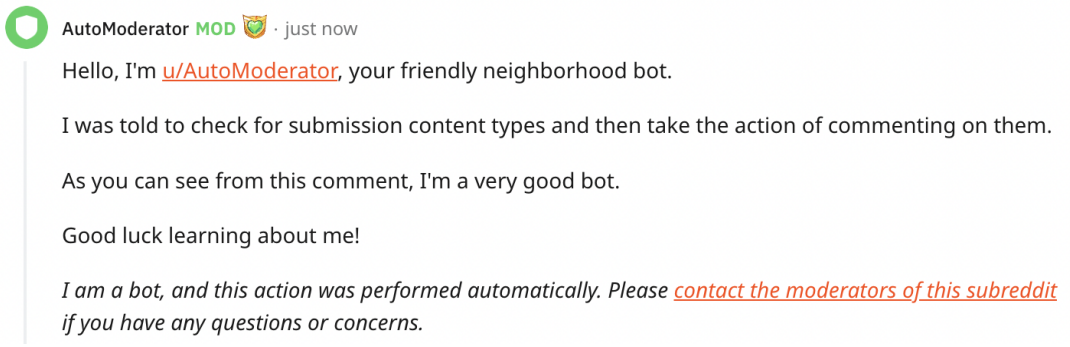
You can read more about what AutoMod can and can't do in r/AutoModerator's FAQ.

AutoMod Rule Structure and Crafting

In the next few lessons, we'll cover basic AutoMod rule structure and how to craft rules. We will limit the focus of the rules in this section to help keep things simple as you get your feet wet. Having a strong foundational knowledge of how AutoMod works will help as you learn how to set up more complex automation tasks in the future. You can also reference further information on writing basic rules in this Reddit wiki page.

Checks and Actions

Each AutoMod rule has a basic structure that consists of *at least* two parts. Those parts are called a **check** and an **action**. You can have more than one check and more than one action in a rule, but you can not craft a rule without at least one of each. Here, you can see AutoMod telling you about what it was checking and what action it took:



If you're familiar with the idea of 'If This Then That', then you already have a basic understanding of how AutoMod works. **If** it finds the things it is checking for **then** it will take the defined actions. We're going to look at a common rule set up that consists of three things: Type, Check, and Action.

- **Type**: Type is a **check** where you tell Automod what **type** of content the rule will be checking. This might be comments, submissions (posts), specific types of submissions, or all types of content. If you do not define a **type**, AutoMod will look at all content submitted to your community, including comments.
    - Although this is a kind of **check**, for the initial purpose of helping demystify AutoMod rule crafting, we'll sometimes refer to it separately from other kinds of checks.

- **Check**: Here, you'll define exactly what Automod is **check**ing for in the **type** of content you're having Automod review for this rule. There are many different kinds of checks AutoMod can perform.
- **Action**: In the **action** portion of your rule, you're telling Automod exactly what it should do once it has found what it was **check**ing for in the **type** of content you defined for the rule. As with checks, there are many different types of actions AutoMod can perform.

If that doesn't fully make sense to you yet, don't worry. We're going to walk through a few examples of how you can use these different rule elements to craft your own simple AutoMod rules before we dive into anything heavy.

A Basic AutoMod Rule

Let's say we want AutoMod to **filter** (for review in mod queue) all posts in our community that have the word "waffle" in the text of the body of the post, then we'll need to create a rule that defines the content **type** as 'submission', **check**s the body field of the post for "waffle" and then takes the **action** of filtering the post if it finds that word. In practice, this rule would look like:

```
#1 Filter posts containing the word waffle

type: submission
body: "waffle"
action: filter
```

In this example, 'type' defines the **type** of content AutoMod will be looking at, or **check**ing. 'Body' defines which field we are **check**ing and "waffle" defines what we are **check**ing for in the body field of the submission. 'Action' defines what **action** AutoMod will take on a piece of content when our checks (the type and field specifications) are met. Because this rule requires there be text in a post to be able to check the body, it will only check submissions that include text.

Naming your AutoMod rules

Above the rule, you'll see a **hashtag (#)** along with a description of the rule. With few exceptions, anything that you place after a hashtag in your AutoMod config will be ignored by AutoMod, making it a handy way to name and describe your rules within the editor. Naming your rules will help *future you* understand what *past you* was doing, as well as keep things simpler for your fellow moderators. Trust us. Capitalizing the names

and descriptions of your rules can also help them stand out from the crowd and keep things organized.

*Quick tip: Although not always appropriate, it's often a best practice to choose '**filter'** as a rule '**action'** instead of '**remove'**. This flags the content for human review in the main mod queue and can help minimize false positives based on AutoMod's inability to review for context. Setting the 'remove' action will send the content directly to your spam queue.*

Reverse match check

You can do a quick change to that basic rule and have it look for all posts that do **not** contain the word "waffle" and filter them by adding a tilde (~) ahead of the field check. That would look something like this:

```
#1 Filter posts not containing the word waffle

type: submission
~body: "waffle"
action: filter
```

A simple but powerful change to a very basic rule. Now any post to your community that doesn't have the word "waffle" would be filtered and put into your mod queue for review.

'Action' vs actions

Before we go further, note that when we reference 'actions' we're talking about the portions of a rule that take actions. Although our action in this lesson's example is literally called 'action', not all actions will be called 'action' when you are writing a rule. You might tell AutoMod to 'set_flair', a type of action it can take, and then define what that flair would be, which would also be a type of action. You can see more examples of actions you can configure AutoMod to take in the full AutoModerator documentation.

Adding a Second Action to a Rule

Now, what if we wanted to track the reason the content was filtered? We can expand upon our original rule by adding another **action**. In this instance, we'll want to use

‘action_reason’, which will tell AutoMod to put a defined reason for the action taken in our moderation log (this will also show in your mod queue when using the ‘filter’ action).

That updated rule, including the rule description after the #, can be seen here:

```
#1 Filter posts containing the word waffle

type: submission
body: “waffle”
action: filter
action_reason: Rule 1.
```

It’s good practice to always include action_reason in your AutoMod rules for transparency and ease of reviewing AutoMod actions. Your action reason should reference the community rule that the AutoMod rule is acting upon. Adding an action reason can also help your team refine your rules over time and reduce false positives, lessening your moderation load in the long run.

Another reason to always set an action reason is ModSupportBot’s AutoModerator Audit feature. This feature allows you to send a specific message to u/modsupportbot with your community’s name in the subject line and receive a data-driven report about your most frequently used AutoModerator rules in return. This bot specifically looks at your action reasons when crafting the report. You can read more about this here.

Adding Multiple AutoMod Rules

If we wanted to add a second rule to our AutoMod config, we’d need to separate the rules in a way that AutoMod understands. To do this, you’ll need to place 3 hyphens between each rule. ‘---’ Make sure there are no spaces before the hyphens and no quotes or any other text elements on the line with the hyphens.

Below, you can see our existing rule first, slightly modified, and a new example listed just below it.

```
#1 Filter posts containing the words ‘scandalous, waffle, or slather’

type: submission
body: [“scandalous”, “waffle”, “slather”]
action: filter
action_reason: Rule 1.
```

```
---

#2 No long content allowed

body_longer_than: 50
action: filter
action_reason: Rule 3 - Our community keeps it brief.
```

We've adjusted our first rule, #1, to now check for more than one word in the body of the post. Here, we defined a simple check list by placing the keywords we want to have AutoMod check within brackets ([ ]) and separating them with commas. Now, if "scandalous", "waffle", or "slather" is found in the body of the post, the post will be filtered and put in your mod queue for review.

In the new example, #2, AutoMod is being instructed to filter (for review in your mod queue) any submitted content, including comments, that are more than 50 characters long. The reason AutoMod is looking at all content types is that no **type** check has been defined and as stated previously, that is the default type setting.

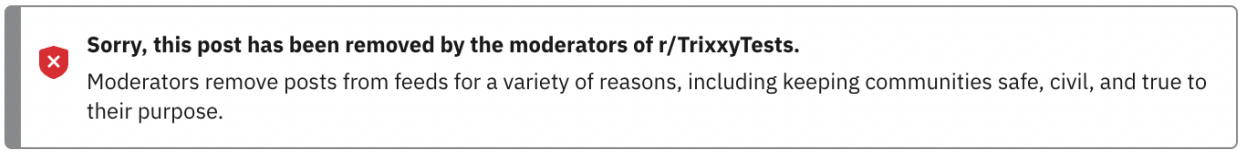
You can also see that we've separated the two rules with '---' and named the second rule by using the #, which tells AutoMod to ignore anything placed after it on the same line.

Quick tip 1: In your AutoModerator configuration wiki page, maintaining a list of all your rules at the top, using # to start each line so AutoMod ignores it can help you when editing your rules. It enables you to see at a glance which rules your community has for AutoMod and also roughly how far down you may need to look to find the one you're after. Using Ctrl+F is also very handy for locating the right rule, especially if you are also giving your rules names using the same # comment method.

Quick tip 2: You can define the order that AutoMod prioritizes different rules with the 'priority' check. This check is always set with a number. Even with a priority set, AutoMod will always first check rules that can lead to content removal (action: remove, filter, or spam) first. You can read more in AutoMod's full documentation.

What a Submitter Sees - Removals and Filtered Posts

A submitter will see this message when viewing the post once Automod has acted on rule #1:



A submitter will see this message when viewing the post once Automod has acted on rule #2:

Those aren't terribly informative messages to see when you've just had your content removed, which is why you may want to use AutoMod to help educate your members when filtering or removing posts and comments via AutoMod rules. The more information you can automagically give to redditors impacted by automation actions, the less questions about removals you'll have to handle in modmail or elsewhere later.

In the next lesson, we'll try using what we've learned to set up a rule that will action content while also helping to educate community members.

Exercise: Writing an AutoMod Rule

Ok, so we now know that we can tell AutoMod what to check for and what actions to take when it finds what it's checking for and we understand how to do that in a basic way. Now you're going to try leveraging what you've learned by crafting a simple rule with multiple actions.

In this exercise, use the full AutoMod documentation as a reference to find the necessary checks and actions, build a brand new rule, and try it out **in your test community**. Use the links provided in the steps below to help find exactly what you should be looking for. **Do not test this rule in an active community**, as it will negatively impact your community members.

Write an AutoMod rule that will do the following:

- Look at all posts submitted in your community.
- Remove posts that are shorter than 30 characters.
- Include a reason for the action to be displayed in your mod log.
- Have AutoMod comment on the post, giving an explanation for the removal. This comment can include markdown and can link to other resources as needed.

When you have your rule structure defined, be sure to also name your new AutoMod rule and separate the rule from other rules before saving. You'll be prompted to leave a reason for the revision when saving your edits to AutoMod. Get in the habit of leaving a reason, as this can be helpful if you or your mod team ever need to review revisions and revert to a previous rule set.

If you run into an error when saving, double check to make sure everything is correct and edit as needed before attempting to save again. Go back and review examples from the previous lessons if needed.

Once you have the new rule saved in your test community, use a **non-mod** alt test account to make a post consisting of less than 30 characters in your test community. It is important that the account posting not be a moderator of the test community, as **AutoMod will not action moderator content** unless the rule it is acting on has been told to overwrite this default behavior. (See 'moderators_exempt' in the full documentation for more information.)

If the rule is written correctly, the post will receive a comment response from AutoMod and will now be in your spam queue, as it will have been removed. You should also be able to review the reason AutoMod took the action in your moderation log. First, check the experience from the perspective of your non-mod test account, and then check things with your main mod account.

On your non-mod account:

- Did you see the banner on the post confirming it had been removed?
- Did you see the comment left on the post?

On your mod account:

- Did you find the content in your spam queue? If not, refresh the queue and check again.
- Were you able to see the action reason for the removal in your moderation log?

Hopefully, you can say yes to all those questions!

If not, that's ok. Let's go over what this rule would look like:

```
# Remove posts comprised of less than 30 characters

type: submission
body_shorter_than: 30
action: remove
action_reason: Rule 1
comment: This post was removed for breaking community rule 1.
```

Although not shown in the example, remember that if you had a rule above this, you'd **need** 3 hyphens to separate the rules. For the sake of simplicity, we've also left a very basic comment and have not shown any use of markdown in the comment. You should generally add more details in your comments and use markdown to add links for further context when referencing things such as rules. We'll go over formatting longer comments later in the course.

If you had success writing this AutoMod rule, great! If it didn't work for you, try going through the above rule and recreating it in your test community's AutoMod config page from scratch for further testing. Once you're done testing, move forward in the course where we'll look at slightly more complicated AutoMod rule crafting.

Quick tip: AutoMod attempts to respect existing mod actions. If you have already approved content, AutoMod should not remove that content, even if it breaks a defined AutoMod rule.

AutoMod Basics Review

Before we get started on writing a slightly more complex rule than in the previous exercise, we're going to review what we know and then discuss **modifiers**.

What do we know so far?

- AutoMod rules must contain at least one check and one action, but can contain more than one of each.
- A type is a kind of check that you define when you want to narrow the kind of content an AutoMod rule is going to look at.

- Generally anything placed after a hashtag (#) in your Automod config wiki will be ignored by AutoMod - these are known as ‘comments’ and are often used to name, describe, and help organize rules.
- You must use 3 hyphens (---) with no space in front of them or other characters on the same line when separating rules.
- It’s good practice to always include an action_reason in your rules.
- You can put a tilde (~) in front of a search check to reverse the check’s behavior.
- AutoMod prioritizes rules that may remove content ahead of their natural order within the config page. You can set priority for rules but it will not overwrite this behavior.
- You can use brackets and comma separators to define simple lists for checks.

If any of this is surprising to you, try reviewing the previous lessons before moving on in the course. If you’re feeling good about what you understand so far, move on to the next lesson.

Modifiers and More

Building a More Nuanced AutoMod Rule

In our very first AutoMod rule example, we told AutoMod to look for the word “waffle” in the body of a post and if found, filter the post. Think of it as a simple way to deal with an influx of people spamming posts with the word “waffle” in them. Childish, annoying, and easily dealt with, right? Alas, although easy to set up and effective, there’s actually a potential problem with this solution!

Matching modifiers change the way AutoMod’s search check behaves. When we search in the body field, there is a default behavior for that kind of search check. The default matching modifier is includes-word if none is defined. This looks for exact word matches in the targeted text. This means that in our original rule, AutoMod will only look for exact matches for the word “waffle”. But what if people just start adding things to the beginning or end of the word to get around the rule or start spelling it “woffle”? It’s not taking into account things like “waffles” or “woffle” or “wafflemonster”. It’s also only checking the body field and not the title field.

We can help define things more clearly for AutoMod by using a matching modifier in our search check. As a reminder, here is our first rule with an action_reason defined:

```
#1 Filter posts containing the word waffle

type: submission
body: “waffle”
action: filter
action_reason: Rule 1.
```

Note: This is intended as an example to help you understand how matching modifiers work. There are better solutions to this kind of issue that also involve modifiers, which we will cover later in the course.

Now, before we add this matching modifier, we’re going to build upon this rule and adjust it in a few ways to make it more powerful and more efficient. First, we’ll combine two checks into a single check.

Combine Checks

You can combine search checks with a plus (+) so that AutoMod looks at multiple surface areas in a single check. After adjusting this rule to check both the title and the body fields, it will take an action if it finds “waffle” in either place. That rule looks like this:

```
#1 Filter posts containing the word waffle

type: submission
title+body: “waffle”
action: filter
action_reason: Rule 1.
```

Match Modifier and Simple List Checks

Of course, now if someone submits a post with “waffle” in the title *or* body of the post, it’s going to be filtered, but it will not be filtered if they use “wafflemonster” or “woffle”. We want to expand what AutoMod acts upon.

So, we’ll need to modify the search check to make what we want clear. We do that by placing a matching modifier within parenthesis after ‘title+body’ to check for any word that **includes** “waffle”. We will also add a simple check list to expand the keyword search by adding brackets and comma separators and adding “woffle”. The rule will then look like this:

```
#1 Filter posts beginning with waffle and woffle

type: submission
body+title (includes): [“waffle”, “woffle”]
action: filter
action_reason: Rule 1.
```

Now, “waffle”, “woffle”, or any word that includes those exact sequential characters within the title or body of a post will also trigger AutoMod to filter the post.

Before we added the includes modifier on the combined search check (body+title), the default was includes-word, which will only look for exact string matches. This default behavior is also true for the body field check when it is not part of a combined search check. Some other field checks have different default modifiers applied to them, which you can read about in the full AutoMod documentation.

Now, we can still improve the rule a bit by having AutoMod provide us a little more information.

Quick tip: If you have a long list of items, you can make your list easier to read by listing them vertically using the correct syntax, and this also allows you to #comment on each line to leave notes for yourself or your mod team, as shown here.

Using a Placeholder

What if we want to know exactly which word was matched when reviewing our mod queue? For that, we can use a placeholder in the action_reason, which will tell us which word matched when we look at the filtered post in our mod queue. Our updated rule now looks like this:

```
#1 Filter posts beginning with waffle and woffle

type: submission
body+title (includes): [“waffle”, “woffle”]
action: filter
action_reason: Rule 1 [{{match}}]
```

Now, we have a rule that should target the inclusion of the words that we defined, while also letting us know exactly what action it took and why. For the sake of transparency, you might consider adding a comment or message to let the submitter know their post

has been filtered, why, and what happens next. Speaking of comments, let's review a couple of helpful quick tips and then add an action for commenting that uses multi-line formatting.

*Quick tip: In Mod Certification 201, we explained that **Toolbox** is a 3rd party extension for desktop browsers created by moderators to extend the functionality of our current moderator tools. If your moderation team is using Toolbox, the **square brackets ([])** around the **{{match}}** placeholder enable Toolbox to highlight the matched item within the content you are reviewing in your mod queue, as we noted in the 'Reviewing your mod queue' section of the 201 course.*

*These brackets are not required if you don't use Toolbox but are still highly recommended, as they also work best with u/**ModSupportBot**'s AutoModerator Audit function that we mentioned earlier when adding an action reason.*

Multi-line Comments

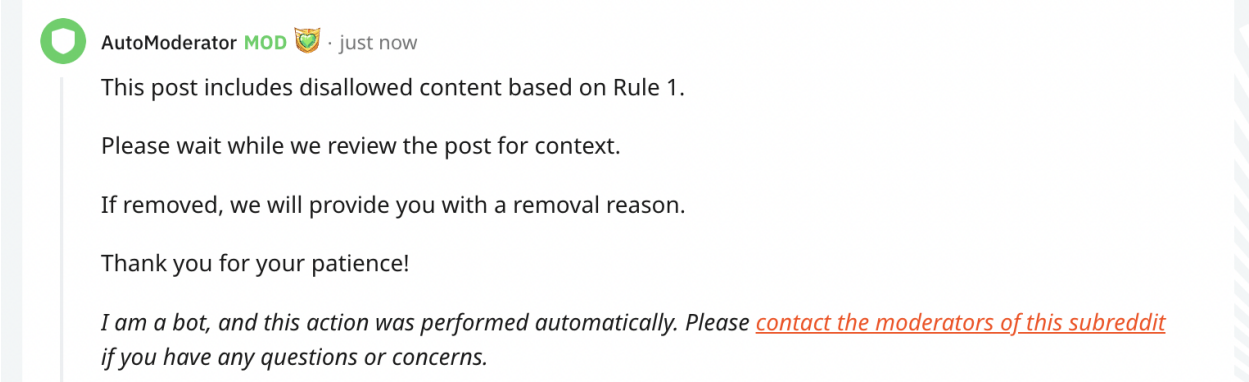
In our AutoMod rule writing exercise, we had AutoMod leave a basic explanatory comment. Now, we're going to learn how to format a comment so that it isn't a single line or wall of text. Formatting your comments, especially longer ones, makes them more legible and neat. Comments that are easier to read are naturally more likely to be read and understood by the intended audience.

You'll format your comment by setting multi-line strings after the comment action. You'll need a single pipe character (|) on the first line after 'comment:' and will need identical space (not tab) indentations for each line of the comment. The rule should look something like this when complete:

```
#1 Filter posts beginning with waffle and woffle

type: submission
body+title (includes): ["waffle", "woffle"]
action: filter
action_reason: Includes disallowed word [{{match}}]
comment: |
    This post includes disallowed content based on Rule 1.

    Please wait while we review the post for context.

    If removed, we will provide you with a removal reason.
```

```
Thank you for your patience!
```

This rule will trigger the multi-line comment action and AutoMod will leave a comment on the filtered post that will look like this:



Remember, you can also use markdown in your comments to help provide your community members with resources to help further understand AutoMod's actions and your community's rules. For the sake of simplicity, we have not used any markdown in our rules.

Quick tip: Consider the 'voice' of AutoModerator in your automated messages and comments. It can aid AutoMod's reception if they're written to be friendly rather than robotic.

RegEx Modifier

We've gone over how a match modifier works in our rule example where we looked for the simple list of keywords "waffle" and "woffle" included in words. In reality, that's not a very effective way to stop someone who is determined to spam some form of "waffle" in your community. Chances are, they'll try all kinds of different tactics to get around any simpler rules you have in place. To minimize the impact of this through automation, you're going to need to know a bit about RegEx, which we'll be covering in our next section.

RegEx

Intro to RegEx

Before we dive into the topic of RegEx, also known as regular expressions, know that you will not be expected to master the use of RegEx by the end of this course. Our intention from this point up to the end of this course is to provide further foundational knowledge and resources that will help you build upon what you've learned thus far.

Understanding the basics of how AutoMod works and how to craft a simple AutoMod rule, as well as where to find resources that will help you do more, is a great foundation for automating tasks in your community. However, having some familiarity with RegEx will help you understand what is truly possible when you are ready to access more complex resources and leverage its power. So, don’t stress if it seems like a lot. Keep going!

About RegEx

One of the most common uses of AutoMod is a simple keyword filter for problematic words or other uses, such as setting keyword-based flair. If you’re looking to filter certain keywords, you have to take into account that users often misspell words or use alternative spellings for a variety of reasons. Evading AutoMod filters via alternate spellings can be a pesky problem. That’s where RegEx comes in.

What is RegEx?

RegEx, which is short for regular expressions, are special strings of characters used to define a search pattern. With AutoMod, RegEx is often used to find keyword variations for rules using the regex modifier alongside a matching modifier. Regular expressions can be simple or incredibly complex based on the text strings being matched.

Why is RegEx important?

If we need to update or manage keywords in an AutoMod rule, it’s easier to update one RegEx string than a list of 20 keywords that are all different spelling variations of the same word. Your filter might need to incorporate many such search terms, making your code block unwieldy and hard to edit.

Regex can be helpful in flagging rule-breaking keywords such as racist or violent terminology. Bad actors may employ alternate spellings to get around such mitigation measures and will often find it easy to evade keyword filters if static keywords are used. Regex is more dynamic, allowing you to anticipate, account for, and prevent these spelling variations and evasion attempts.

Common examples include replacing letters in naughty words with asterisks or using an @ symbol in place of the letter ‘a’. Using RegEx, we can better adjust for these variations and catch a far wider range of keyword spellings than a simple string of static keywords allows.

That said, there’s also no need to recreate the wheel! The Standard Library on r/Automoderator has rules that you can incorporate into your code and the Profanity Filter rule provides a great example of RegEx patterns. Running a search in r/AutoModerator can also help you find at least a rule you can edit and build on instead of starting from scratch. Though it would still need adapting to suit your community's specific needs.

RegEx Example and Explanation

Despite our love of breakfast foods, we’re going to move away from the “waffle” keyword example in this lesson. Instead, let’s say we wanted to use AutoMod to catch the word ‘yourselves’. There are a large variety of ways users might type the term including ‘urselves’, or even ‘yourselfs’. They might use the singular instead of the plural. In an AutoMod ‘rule’ we could list every conceivable spelling as an independent search term, but RegEx allows us a way to capture many spellings with one bit of code.

Below, instead of listing each spelling as a separate term, we've created the following RegEx to capture many possible versions of “yourself” / “yourselves”.

```
(yo)?u(rsel[fv](e?s)?)
```

You’re probably wondering what you’re even looking at right now and that’s ok. We’ll do a quick walk through to explain the characters used in this example. We also recommend opening up this wiki for additional context but don’t expect to immediately be a RegEx wizard. That will take a lot of time and practice!

(yo)?u(rsel[fv](e?s)?) breakdown

Think of the characters within **parenthesis** as groupings (called **capture groups**) that are treated as a single item. Anything directly before the **question marks** is optional when matching for a string, including groupings. The **square brackets** can be thought of as ‘or’ - as in, this search string must include this **or** that for it to match. Because AutoMod is case insensitive, we do not have to specify for capital letters to catch those here.

Knowing that, let’s look at this RegEx example again.

(yo)?u(rsel[fv](e?s)?)

First, we have the optional ‘(yo)’ capture group. Why is this optional? Because the question mark after it defines it as such. Next is the required literal ‘u’. At this point, we know that the word can start with ‘you’ but **must** at least start with ‘u’.

Next, we’ll break down the more complex grouping, which includes a subgrouping, into parts. ‘(rsel[fv]’ is a required part of the grouping. We require ‘rsel’ to at least follow the ‘u’ for the string to match and because of the square bracketed portion ‘[fv]’, ‘rsel’ must also be grouped with an f **or** v following it for the string to match. Now, we know that we can match yourself, yourselv, urself, or urselv.

At this point in the RegEx, if you were using the matching modifier **includes** in your AutoMod rule, you could actually stop and have RegEx that would also catch plurals because it would look for **(yo)?u(rself[fv])** inside of other words, meaning it would catch “blurselfp” or “yourselves”. While using the includes modifier, you could even just have **‘u(rself[fv])’** since you wouldn’t actually need to define the “yo” for it to match words starting with those letters. That said, the default matching behavior for some AutoMod field checks and all combined search checks is **includes-word**, which looks for exact word matches. If you stopped at this point without defining the matching modifier of **includes**, it would not catch the variations that ended in “es” or “s”.

Going back to our RegEx example, the e and s subgrouping after [fv] is listed as (e?s) and is followed up with ?) to close the grouping that began with ‘(rsel[fv]’. Because the ‘e’ is followed by a question mark in its subgrouping, it’s optional. And because the entire subgrouping of (e?s) is also made optional by the question mark directly after it, this means the word could include both the e and the s at the end, just the s at the end,

or neither of them at the end and it would still be a match as long as those characters are directly preceded by either urself, urselv, yourself, or yourselv.

Now, say we want to use this in an AutoMod rule. The example below would check the body and title fields of posts for keyword variations that are an exact string match for a stand alone word. If found, it would filter them into our mod queue, displaying the exact match it found that triggered the filtering.

```
# Filter posts including 'yourself'

type: submission
body+title (regex): (yo)?u(rsel[fv](e?s)?)
action: filter
action_reason: rule 2 - [{{match}}]
```

If instead we wanted to have AutoMod look for a match of the keyword variations inside of words, we could adjust the RegEx and add the matching modifier ‘includes’, as shown in the rule below:

```
# Filter posts including 'yourself'

type: submission
body+title (includes, regex): 'u(rsel[fv])'
action: filter
action_reason: rule 2 - [{{match}}]
```

We could also use list checks with RegEx if we wanted to search for a number of variations of multiple keywords in a single rule. We’re not going to dive into that here, but knowing this can help you craft more efficient rules in the future.

If you’re feeling a little lost, stick with us. This topic has a very real learning curve and a single basic RegEx example broken down for you isn’t going to mean you can run out and easily write your own RegEx. What we’ve covered in this example is just a tiny view into what RegEx is capable of when used in an AutoMod rule. However, you should now have a basic understanding of what RegEx can do, how it might do it, and how you can include it in an AutoMod rule.

We’ll provide a list of some resources that will help you learn more about RegEx and writing and testing regular expressions at the end of this section of the course. These resources are a paired down version of what we included at the beginning of this course.

Next, using what we’ve just learned, let’s try to write a simple RegEx from scratch.

RegEx Action Item

Let's say you need to prevent personal information from being shared in your community, specifically someone's name. The name is spelled Susan, but you've noticed that users are spelling it in different ways. It's not sufficient to filter just the name 'Susan', you need to account for nicknames such as 'Sue', 'Suzy', 'Susie', and even 'Suzan'. You've also noticed that users are spelling it as 'Suzanne' or "Susanna / Suzannah" so you'll need to take that into account as well. The standard library also provides pre-written rules for dox detection that can be very helpful for general doxx detection and even detecting username mentions.

Before you get started, take a look at the example here on parenthesis to understand how you can define separate capture groups within a single set of parentheses using pipes '|' as dividers within the capture group. Using this character can be very helpful when writing the RegEx to capture all of the string matches we'll be searching for below.

Now, assuming you're using the default match setting (**includes-word**) in your AutoMod rule, try writing a RegEx that takes into account all of these various spellings:

- Su
- Sue
- Susan
- Suzan
- Suzy
- Susy
- Suzie
- Susie
- Susanna
- Suzanna
- Susannah
- Suzannah

You don't need to write an AutoMod rule. Just try to apply what we've talked about in this section and see what you come up with. Use this link for help writing the RegEx and this link to test it.

Take your time, and when you're ready to check what you've come up with against the solution, click next.

RegEx Action Item Solution

Answer: sue?[sz]?(y|ie|ann?)?(e|a)?h?

This solution will capture every spelling variation for Susan that was in our list. If you were to take that RegEx, using it to find and filter for keyword variations in the body and title of submissions, setting up an AutoMod rule with no matching modifier, it may look like this:

```
# Filter posts for sue and variations

type: submission
body+title (regex): 'sue?[sz]?(y|ie|ann?)?(e|a)?h?'
action: filter
action_reason: rule 3 - [{{match}}]
```

Note that we only use the (regex) modifier on the combined field (body+title) check, making the match behavior default to ‘includes-word’. If we had added the ‘includes’ matching modifier, this RegEx would capture any word with “su" in it, creating a **ton** of false positives in our moderation queue. For this reason, it's important to test your RegEx with testing sites and sometimes in a test community, posting and commenting with terms you don't want to match as well as those you do, to ensure you’re matching strings as intended. Imagine every post with "assume" or "issue" or "usual" ending up in your mod queue. Not fun.

Also note that in our example rule, we’ve included the [{{match}}] placeholder within our action_reason. This tells AutoMod to display exactly which variation of the word triggered the rule and will display that information in your mod log and your mod queue. Extremely helpful both in seeing what you're catching and also catching and approving false positives.

So, did your answer match the above RegEx? If not, did you test it out using the provided link and review what it matched? If you can answer yes to either of those, that’s great. You don’t need a perfect answer to make progress here and testing the RegEx to see how it works is a great first step. If you were unable to answer yes to

either, compare what you have with the above and try testing it and seeing what you might want to do differently.

If it's not perfect, don't worry. RegEx can take a lot of time and effort to truly understand, and we recommend regularly reviewing resources to help you on your way. Below, we've shared a few you can reference as needed. You might want to go ahead and bookmark them for future use before moving on to our next section on common AutoMod rules.

Resources

- Regexone.com | Good for learning RegEx
- Regex101.com | Good for testing RegEx
- Regexr.com | Good for learning and testing RegEx
- 001Guy001's Regex Primer | Easy to understand RegEx explainer

AutoMod Rule Examples

Common AutoMod Rule Examples

Hopefully at this point in the course, you're feeling like you understand how AutoMod works and have at least a basic understanding of what RegEx is and how it can be used to make AutoMod rules more powerful. Now, we're going to slow down and just look at some examples of the many varied ways Reddit moderators use AutoMod in their communities. Looking over this section is optional but recommended.

There are many AutoMod rules or instructions already written, and you can find some of them in r/AutoModerator's library, and r/ModGuide's list of snippets. Taking a bit of time to review existing rules while referencing AutoMod's full documentation and other AutoMod resources can help you gain a deeper understanding of AutoMod rule crafting.

We'll be sharing some examples in the next couple of lessons. You won't walk away from these examples understanding how to do the same from scratch but you should come away from them with a better understanding of what is possible. Let's start!

There are many AutoMod rules or instructions already written, and you can find some of them in r/AutoModerator's library, and r/ModGuide's list of snippets. We'll be sharing some examples in the following lessons.

A couple of things to note: We recommend adding an 'action_reason' for all of your rules, even if they are not displayed in the examples below. We also recommend naming your rules (using the # element) even if examples shown might not always be named.

This rule below removes content that receives 2 (or a number of your choice) or more reports, and sends a modmail so that moderators can check if it was the correct action:

```
# Removes reported content
reports: 2
action: remove
modmail: |
    {{permalink}}
    The above {{kind}} by /u/{{author}} was removed because it received 2
reports. Please investigate and ensure that this action was correct.

---
```

And this one, which has AutoMod leave a sticky comment on every post in the community:

```
   type: submission
    is_edited: false
    author:
        is_moderator: false
    comment:  |
        Your comment goes here

    comment_stickied: true
---
```

Quick tip: Rules that remove content when receiving a set number of reports can be very handy as a safety net. Be aware that any item previously approved by a mod will not be removed by AutoModerator if the content receives new reports after being approved.

Special AutoMod Rule Examples

Sometimes you may need AutoModerator rules that are very specific to your community, that perform a specific task, or are for a specific situation.

Here, we've listed a set of three AutoModerator rules that work together to enforce a rule for question threads where only serious replies are requested.

The first rule posts a sticky comment on each thread that has a specific flair (‘serious replies only’) to remind users how to format their comment:

```
# Serious tag commenter

flair_template_id: YOUR_FLAIR_TEMPLATE_ID
comment: |
      # This post has the serious replies only tag, so please only post serious, on-topic replies.

      **All top-level comments must start with: `Answer:`**

      If you see a comment that is off-topic or not a serious reply please report the comment.

      Posts not suited for the 'serious replies only' tag will be removed.
comment_stickied: true
comment_locked: true

---
```

Looking at the rule above, are you able to spot the checks and actions? Try and find them as you look through the next 2 rules. Here, we’ve **check**ed for specific post flair with flair_template_id. The **action**s we’ve taken are leaving a multi-line comment with comment, stickying the comment with comment_stickied, and locking the comment with comment_locked.

The second rule enforces that comment requirements are followed. It removes every top-level comment that does not begin with ‘Answer:’:

```
# Serious tag 'answer:' comment enforcer

type: comment
is_top_level: true
parent_submission:
      flair_template_id: YOUR_FLAIR_TEMPLATE_ID
~body (starts-with): ["Answer:"]
action: remove
action_reason: "Comment did not include answer: in serious post"
message: "Hi /u/{{author}}, [Your {{kind}}]({{permalink}}) in /r/{{subreddit}}
was removed because your top-level comment in a `Serious replies only` post
didn't start with `Answer:`. To help ensure all comments in `Serious replies
only` posts are actually serious you must start your comment with `Answer:`.
This does not apply to posts without the `Serious replies only` flair or
replies to other comments."

---
```

The third rule sets the post flair on the thread based on matching keywords in the post. This is for when the OP forgot to set their own post flair to request serious replies only before submitting the post. This ensures that threads are flaired correctly. AutoMod also leaves a stickied, locked, comment on the post and messages the OP with information about the set_flair action it took and why.

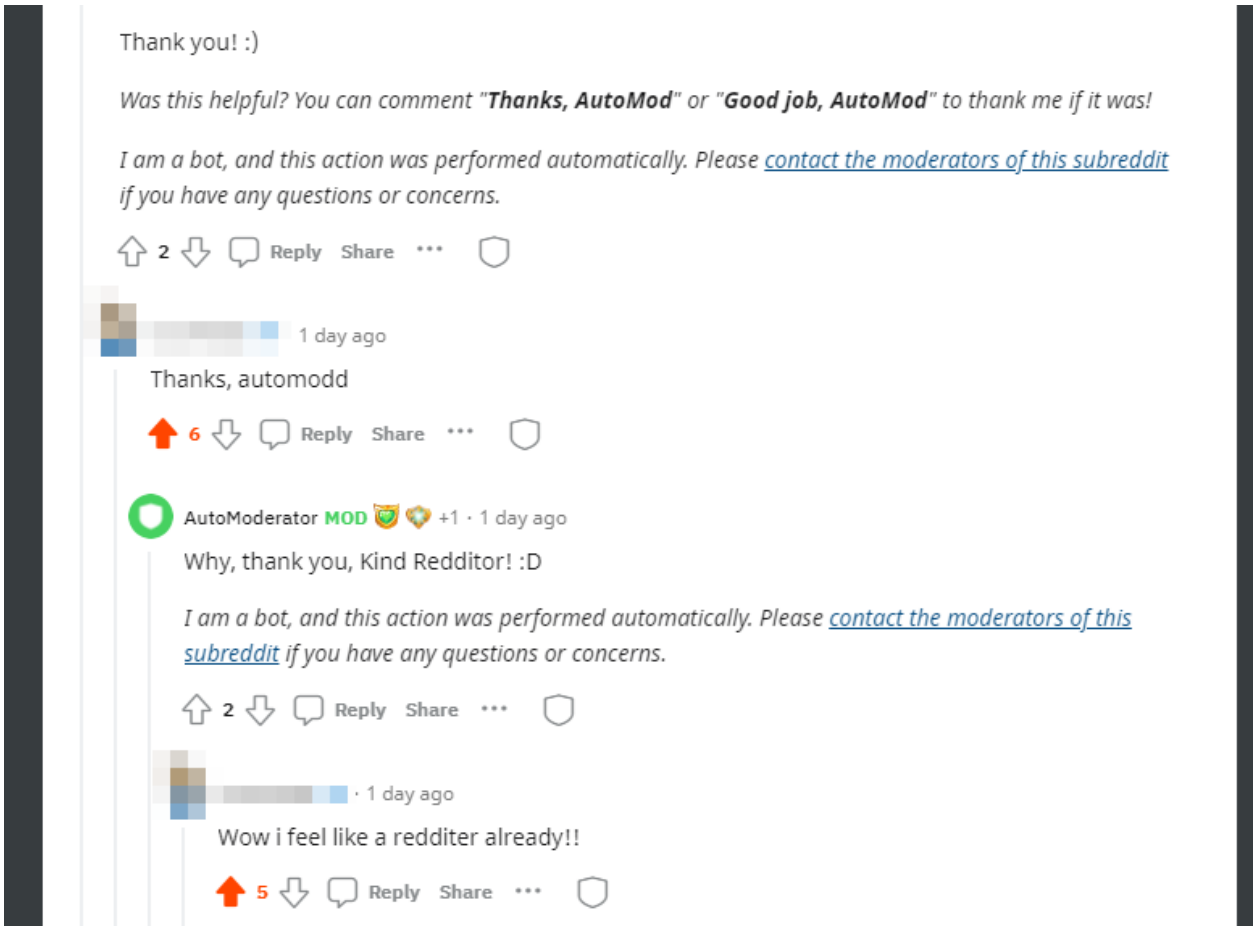
```
# Serious tag auto set from keyword

type: submission
title (RegEx, includes):
      - "(\\(|\\[)(.*)serious(.*)(\\)|\\])"
      - "(only|just|keep)? ?serious ?(answers?|comments?|responses?)( ?only)?"
      - "keep ?(answers?|comments?|responses?|repl(ies|y)) ?serious"
set_flair:
      template_id: "dcdfcd54-3427-11ec-90db-525ee9a526d8"
comment: |
      # This post has the serious replies only tag, so please only post
serious, on-topic replies.

      **All top-level comments must start with: `Answer:`**

      If you see a comment that is off-topic or not a serious reply please
report the comment.

      Posts not suited for the 'serious replies only' tag will be removed.
```

```
message: "[Your Post]({{permalink}}) has been automatically flaired as 'Serious
replies only' due to keywords in the title, if you believe this was a mistake,
reply to this message"
moderators_exempt: false
comment_stickied: true
comment_locked: true
---
```

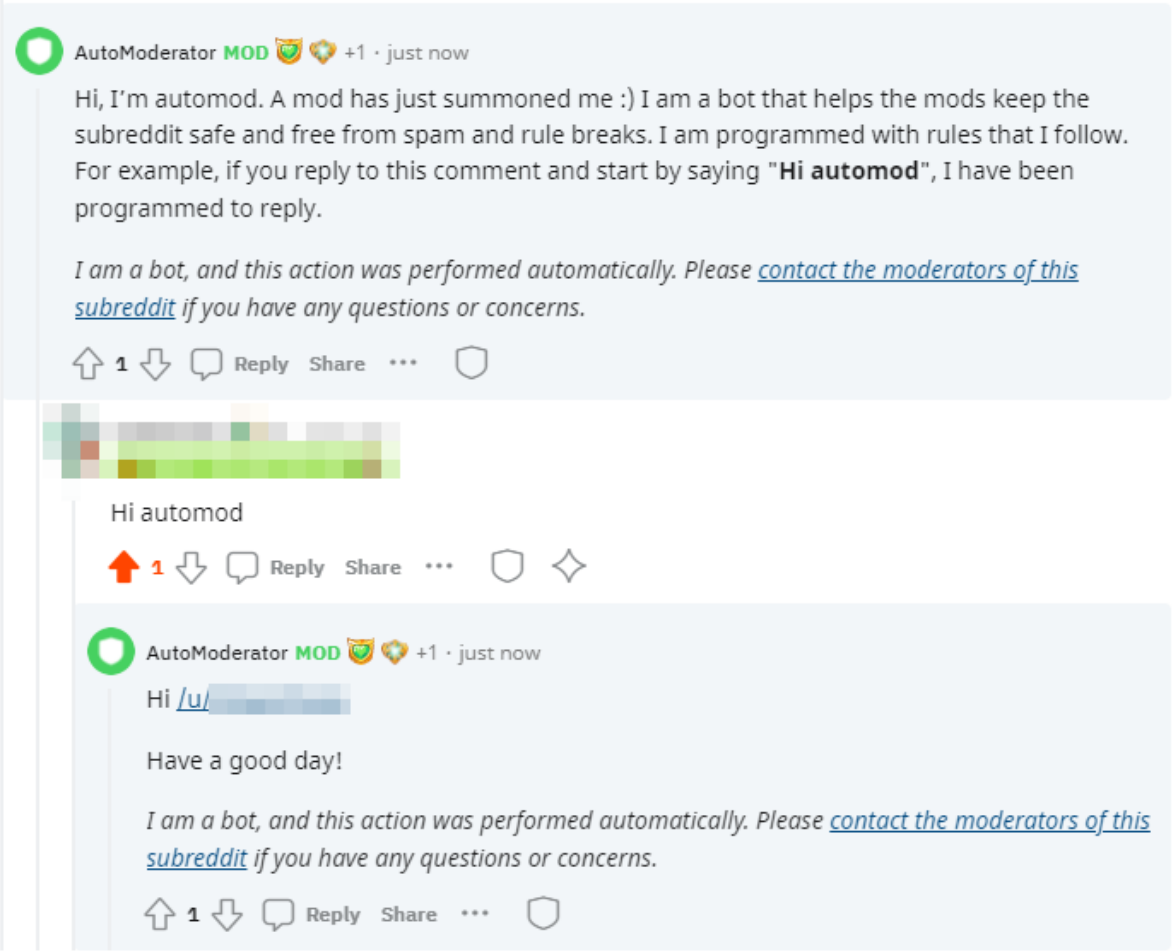
Fun and Educational Examples of AutoMod Usage

AutoModerator and other bots can be used to inform, educate, and amuse. Here, we've shared some screenshots to demonstrate AutoMod's capabilities.

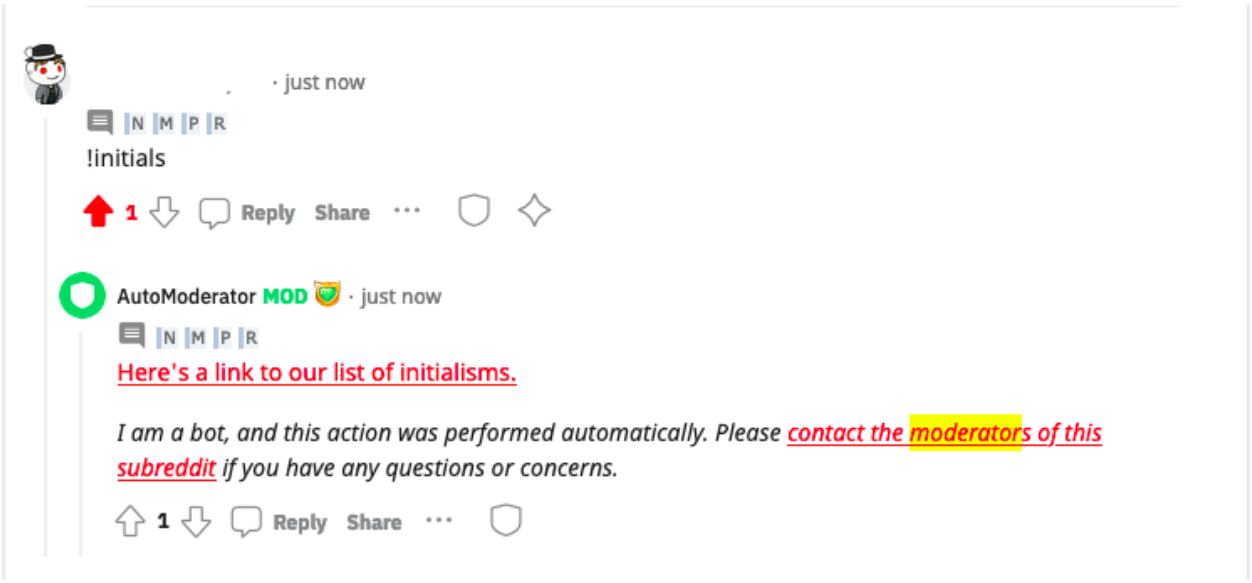
In our first example, AutoMod has been used to welcome a redditor. This also demonstrates how AutoMod can be configured to react to a community member's comment in a friendly and fun way.



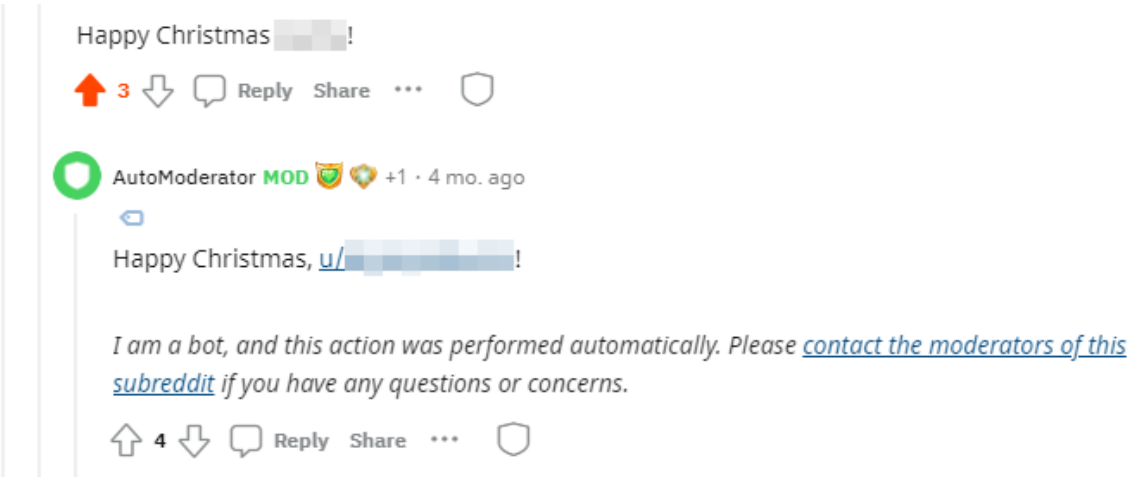
This next screenshot displays a command and response that has been set up to explain and demonstrate what AutoMod can do:



Below, AutoModerator is being used to comment with links to pages in a community's wiki. In this community, important public figures are often referred to by their initials. A list of these initials and their full name equivalent is listed in the wiki and can be summoned. Similar commands can be created for community rules and other important information.



And here AutoMod was set up to look for anyone sharing festive greetings in a number of ways and languages, including a variety of end-of-year celebrations, and to repeat the greeting back to them:



Quick Tip: Having AutoMod leave a well formatted comment any time it actions a piece of content will lower the amount of time you and your mod team spend reaching out and answering questions from new redditors and confused community members.

Mod Bots and Scripts

Mod bots

In this section, we're going to just scratch the surface on mod bots and scripts. We want you to finish this course with an understanding of what is available to you and what is possible and this is a part of that. Now, you're almost at the end of the course. Keep going!

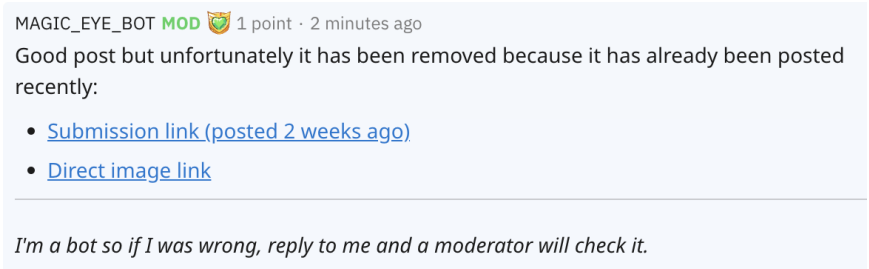
What is a mod bot?

Moderation bots or 'mod bots' are bots just like AutoModerator except they have been created by redditors to aid moderators and are not a native part of Reddit itself.

Mod bots typically take the form of user accounts already set up to run a bot script making it reasonably straightforward for you to add them to your community.

Example of an existing mod bot

For context, u/MAGIC_EYE_BOT is a popular and highly customizable repost detection and moderation bot authored by u/CosmicKeys. This bot can take a variety of actions, depending on how a mod team chooses to configure it. Below is an example of this mod bot commenting on a removed post that was detected to be a repost. Note that the comment includes context and invites the submitter to respond if there is an issue so it can flag that issue to mods, if needed:



Adding a mod bot to your mod team

These bot accounts require you to add them as a moderator for your community with specific permissions, depending on their role, and often also require configuration within

a wiki page. It's wise to only give the mod permissions absolutely required for the bot to perform its intended tasks, and some mod teams also like to keep bots at the bottom of the moderator list as a precaution.

Making a mod bot work in your community

As stated previously, mod bots often require some configuration. How to configure each mod bot that requires configuration will vary. Some respond to instructions written in YAML, similar to AutoModerator, and others respond to other languages such as JSON. Typically, each bot creator will write up some set-up instructions for you to follow, and some bot creators also provide a default configuration setup for their bot that you can copy and paste.

These instructions and help can sometimes be found via the bot's profile page, sometimes as a sticky post. Other times there may be a subreddit of the same name with a sticky post or a wiki page that lists the instructions. Sometimes the bot creator will name a preferred method of contact if you are having trouble but if not, you can try modmailing the bot subreddit, and if there isn't one, message the developer directly.

You can find a list of popular moderation bots here. You might also consider requesting someone write you a bot. You can try r/requestabot for that purpose.

What bots can and can't do

Like AutoMod, moderation bots can handle repetitive work that doesn't necessarily require the attention of a human moderator. Mod bots can also act on past content and connect with outside data sources, unlike AutoModerator. That said, they do need to be hosted somewhere. For some (premade) mod bots, the developer (dev) has done this for you, but if your community should need a custom-made bot (script) you would need to figure out somewhere to host it.

Bots are limited by the Reddit API (application programming interface). The Reddit API is an interface that allows apps and computers to interact directly with the site. Bots and apps can only do things on Reddit that a user or mod can do. However, not everything a user can do can be done by a third-party bot. For example, Reddit chat is not part of the API and as such cannot be accessed by a bot.

Understanding the limitations of the Reddit API will help you better understand what moderation bots can and cannot do. Some newer Reddit features don't yet have full API support.

Are mod bots secure? How do I vet them?

It is good practice to research moderation bots before adding them as moderators. Many of them have open-source code, which you can review to see how each bot works. Some bots are not open-source and rely on the reputation of the developer as a quality contributor to Reddit operating in good faith. You can and should do research on any developer or moderation bot that you plan to use or unsure of. There are dedicated communities such as r/bot or r/botwatch that may help you.

Someone writing their own bot could also incorporate two-factor authentication into the bot script to increase security for the account.

How do I get started writing my own Reddit bot(s)?

Well, you'll need to be prepared to learn quite a bit unless you're already familiar with some programming languages. Reddit bots can be written in any of the popular programming languages as long as it interacts with the Reddit API. The most popular language is Python and the most popular way to work with the API is with Python via a software package called PRAW. Make sure to get familiar with the API access rules and bottiquette as a first step and visit r/RedditDev for more information.

Note that Reddit is currently working on a developer platform that will provide a new way for developers to create bots (and more) for Reddit. The waitlist for the developer platform's beta access can be found here.

These bots we've been talking about? They're made up of script files and can also be referred to as 'scripts'. We'll talk a bit about this next before wrapping up the course!

Scripts

What are scripts?

A script is a special text file that consists of a set of instructions that work within a program on your computer. Most scripts for Reddit are written in the Python

programming language. Think of a script as a way for you to take moderator actions on Reddit directly from your computer without having to use a web browser or an app.

Why do people write scripts for Reddit?

The most common use case for scripts on Reddit is providing capability that Reddit's native tools and systems do not. Automoderator cannot act on a post an hour after it's been posted. A script, however, can target the creation time of the post in order to set a flair, sticky the thread, lock it, or do a variety of other actions. The most widely used mod bots use scripts to do things that AutoModerator cannot do such as detect reposts, enforce link flair on posts (if not assigned when posting), remove posts via flair, or allow community members to vote on the quality of the post.

Used in conjunction with a robust AutoModerator configuration, scripts can extend the range of actions available for automation and act as powerful tools for moderators. And as you already know, automating routine actions can allow moderators to spend more quality time on other aspects of community management.

Can I write a script for my own bot?

There are a wide range of uses for scripts but they do have a high barrier to entry. Besides understanding how to write them and then doing so, you also need to host them and run them yourself on a pc or via a cloud hosting service. Hosting bots can sometimes incur costs as well.

If you do wish to create a script, be prepared to learn! Many people with no previous scripting or programming language experience have gotten started with things like simple bot scripts, so it's more than possible if you're ready to put in the effort.

First, please read through the bottiquette to help ensure you understand community expectations around bot behavior. Ignoring the bottiquette will not win you friends and may even lead to your bot being unwelcome in a number of communities or across Reddit itself. But still, seriously, wow. If you go from this course to writing your own bot scrips, you're probably an incredibly quick study. Possibly a wizard… hrmmm.

Well, that's it! You've almost completed our course on automation for beginners. Let's go ahead and review. Once that's done, you'll be tested on what you've learned.

Automation for Beginners Review and Final

Automation for Beginners Review

You've done it. You've reached the end of Automation for Beginners. Congratulations!

There's one last thing you need to do, and that's take the final! We've provided a quick check list below of what you will be expected to know going into this final. If you feel like you aren't familiar with any of these, be sure to go back and review the related lessons.

You should know:

- What automation is and how it relates to moderating on Reddit
- When and why automation can be useful in community moderation
- When using automation is inadvisable and why
- What AutoMod is
- Where to edit AutoMod rules
- What parts make up an AutoMod rule
- Where to go for help with AutoMod issues
- How to write a simple AutoMod rule
- What AutoMod can and can not do, more or less
- Best practices around basic AutoMod rule crafting
- What RegEx is
- How RegEx can be used with AutoMod
- How matching modifiers behave
- What a mod bot it
- What a script is
- Why mod bots/scripts may be useful in community moderation

Hopefully, you're feeling good about what you've learned in this course and are ready to go.

You'll need to score 80% correct to pass and earn a trophy for completion in this course. Include your username when prompted during the final if you would like to receive a trophy. Note that trophies are granted bi-weekly. You've got this!

Automation for Beginners Final

1. In RegEx, what is a capture group?
    a. A grouping of characters within a set of square brackets [ ], treated as a single item
    b. A grouping of characters within a set of parentheses ( ), treated as a single item
    c. A grouping of characters within a set of parentheses ( ), treated as optional
    d. A grouping of characters within a set of square brackets [ ], treated as optional
2. What is the default matching modifier for a combined search check?
    a. includes
    b. starts-with
    c. includes-word
    d. regex
3. In the following AutoMod rule, what is type: submission and what does it do?

    # Filter posts for sue and variations

    type: submission
    body+title (regex): 'sue?[sz]?(y|ie|ann?)?(e|a)?h?'
    action: filter
    action_reason: rule 3 - [{{match}}]

    a. It is a check that tells AutoMod to check all post types
    b. It is an action that tells AutoMod to act on all submitted content types
    c. It is a check that tells AutoMod to check all submitted content types
    d. It is a check that tells AutoMod to check all submitted comment types
4. Of the following characters, which would you use to separate AutoMod rules?
    a. #
    b. [ ]
    c. :
    d. ---
5. In the following AutoMod rule, what is includes?

    #1 Filter posts beginning with waffle and woffle

    type: submission
    body+title (includes): ["waffle", "woffle"]
    action: filter
    action_reason: Rule 1 [{{match}}]

    a. A matching modifier
    b. A RegEx modifier
    c. An action modifier
    d. A special field only used for Toolbox

6. Of the following, which is true?
    a. AutoMod rules require a defined type check for them to work
    b. AutoMod rules require a defined action reason for them to work
    c. AutoMod rules can have more than one check and more than one action
    d. AutoMod rules must have at least one check but do not require an action
7. If you want to use an existing mod bot in your community, what must you always first do?
    a. Add the mod bot to your moderation team
    b. Write a custom bot script for the mod bot
    c. Find hosting for the mod bot
    d. Request a mod bot from Reddit admins
8. Custom scripts allow bots to check and take action on things that AutoModerator can not.
    a. True
    b. False
9. Which of the options provided below is **not** something AutoMod can do?
    a. Sticky a comment
    b. Require all submissions to include the word 'beefcake'
    c. Act on old or past submissions
    d. Send your mod team a modmail message
    e. Set flair on a post based on a keyword match
10. Which mod action would **not** be best handled through automation?
    a. Responding to modmail messages from the community
    b. Setting required but missing post flair based on keyword matches in a post's body
    c. Removing content that has a large number of user reports
    d. Looking for the inclusion of doxxing keyword match variations in all submissions
11. Why is it wise to add an action_reason to every AutoMod rule?
    a. It can help you know when to refine your rules to reduce false positives
    b. AutoMod needs to know why you're asking it to do something
    c. It shows in the mod log and mod queue
    d. u/ModSupportBot's AutoMod Audit feature looks at action reasons when crafting a report
    e. If you don't add an action reason, the rule won't work
12. What is the correct syntax for a reverse body field check?
    a. bo[dy]:
    b. (body)?:
    c. #body:
    d. ~body:
13. There is more than one way to format a check list of keywords in an AutoMod rule
    a. True
    b. False
14. What is one way you would **not** use # in your AutoMod config page?
    a. Directly in front of a rule's type check, to help it stand out
    b. To create a list of your AutoMod rules at the top of your config page

c. Ahead of a rules' name and description which help you locate the rule, so AutoMod doesn't see it
d. Ahead of text meant to clarify a rule without having AutoMod act upon that text

15. What is the purpose of the question marks in the example below?

```
# Filter posts including 'yourself'

type: submission
body+title (regex): (yo)?u(rsel[fv](e?s)?)
action: filter
action_reason: rule 2 - [{{match}}]
```

a. To make the item following it optional in string matching
b. To make the item preceding it optional in string matching
c. To allow for wildcard letters in its place in string matching
d. To make the item before or after, but not both, be required for string matching

16. If you would like a Reddit profile trophy upon passing this final, please provide your username. Do not include the characters "u/" before your username.

    If you do not want a trophy, please put "no trophy" in the field.

You've Done It!

Congratulations!
You've reached the end of Automation for Beginners. Choose from one of the navigation options below to continue or click 'Next' to see a list of resources for learning more about Automation.
Visit Mod Skill Training Return to Reddit Mod Education

Automation Resources

Here, we've provided a non-exhaustive list of different resources you may find useful while learning about AutoMod and automation. Reference these resources when you want to dive deeper and be sure to bookmark those you find most helpful.

AutoModerator resources

- AutoMod full documentation
- Reddit's Mod Help Center
- r/AutoModerator wiki index
- r/ModGuide's master list of AutoMod resources
- r/ModGuide's guide to using Content Controls in concert with AutoMod
- YAML (the language of AutoMod)
- r/ModSupport thread full of tips

Pre-written AutoMod rules

- r/AutoModerator's library of common rules
- r/ModGuide's AutoMod rule snippets
- r/myautomod's list of Automod rules
- r/buckrowdy's AutoMod rule snippets

RegEx

Learning

- r/myautomod RegEx primer
- RegExOne
- Python.org Regular Expression syntax
- Regular Expressions quick start

Testing

- RegEx101
- RegExr: Learn, Build, & Test RegEx

Mod bot and scripting resources

- r/ModGuide bot guides

- r/ModGuide mod bot list
- r/RequestABot
- r/ModGuide script guides
- r/learnpython
- r/redditdev
- Python.org
- Reddit API documentation
- Reddit API rules
