

Using AbuseFilter extension to combat vandalism on a wiki

[[User:Daimona Eaytoy]]



WIKIMEDIA
FOUNDATION

SWT Indic Workshop Series
6 December 2020

a.k.a.

**How to fight vandalism
without lifting a finger**

(almost)



WIKIMEDIA
FOUNDATION

a.k.a.

...

VANDAL

WARS

**A long time ago in a wiki far,
far away....**

*People were faced
with a lot of blatant
vandalism. Poor patrollers
had to spend their time
deleting easily-predictable
badwords, taking time away
from the important things.
They needed some tool
to do this for them.*

What are we going to need?

Magic.



Except it's this

Kind of Magic

Extension: AbuseFilter *

No, it doesn't have a logo ٩(ツ)ノ



* a.k.a. “edit filter” on enwiki

How does it work exactly?

- You write a rule in a simple scripting language, that will be checked for each edit (1);
- You specify a set of actions (“consequences”) to be taken if the rule matches;
- Rinse and repeat as many times as you want (2).

(1) and also other actions

(2) there's a limit, but it's hard to reach



You need some privileges to do this. On most wikis this is restricted to administrators, sometimes there's a specific group of AbuseFilter editors

Rules



*This is not an introduction to
programming languages*

For that, see

**THE
C
PROGRAMMING
LANGUAGE**

Brian W. Kernighan • Dennis M. Ritchie

PRENTICE HALL SOFTWARE SERIES

Language

- Language is easy to learn and to read
- Quite powerful, but nowhere near a “full” programming language (for the good and the bad parts)
- **Pro tip:** the language is not Turing-complete

What rules looks like

```
1 !("confirmed" in user_groups) &
2 page_namespace == 0 &
3 (
4   new_size < 50 & old_size > 300 |
5   new_size/(old_size + 1) < 0.1
6 ) &
7 !(lcase(new_wikitext) rlike "#\s*redirect|{{(?:db-(?:attack|g10)|wi|wiktionary\s*redirect)\s*[|]}")
8 |
```

Ingredients

Like human languages have nouns, verbs, etc., programming languages have some components:

- Literals (42, 'force', true, ...)
- Operators (+, -, &, |, ==, ==, !, ...)
- Keywords (in, like, ...)
- Functions (contains_any, count, substr, ...)
- Built-in variables (new_wikitext, user_name, page_title, ...)

Data types

Some components of the language can have a type. Intuitively, 42 has the type “number”, and “I am your father” has the type “string”.

This is also true in programming languages, with some additional types.

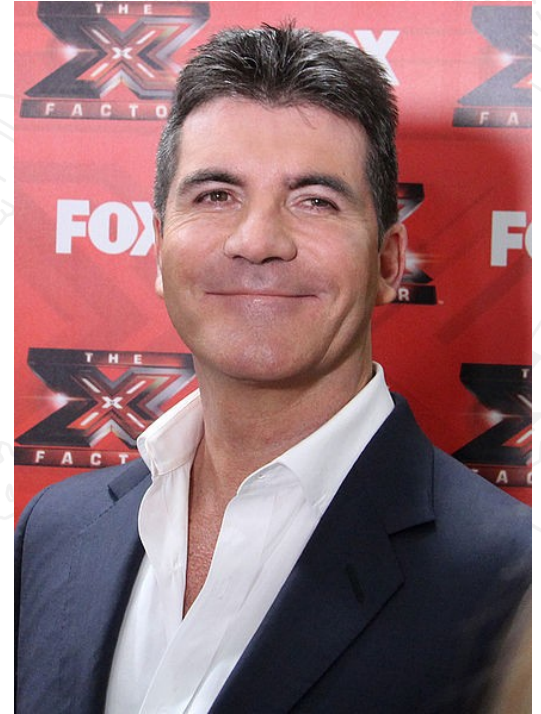
Data types

- Integers: 0, 1, -67, 42
- Floats (“decimals”): 0.1, 3.14, -51.798
- Strings: "foo", 'bar' (can use single or double quotes)
- null (means something is empty)
- Boolean: true and false (truth values)
- Array: ['foo', 3, false] (list of values of any type)

Casting



39 / 100



From left to right:

- [\[\[User:Ab5602\]\]](#), CC-BY-SA-3, via Commons
- [\[\[User:Maplestrip\]\]](#), CC-BY-SA-4, via Commons; writing added by myself
- Alison Martin of SimonCowellOnline.com, CC-BY-SA-2, via Commons

Casting



From left to right:

- [\[\[User:Ab5602\]\]](#), CC-BY-SA-3, via Commons
- [\[\[User:Maplestrip\]\]](#), CC-BY-SA-4, via Commons; writing added by myself
- Alison Martin of [SimonCowellOnline.com](#), CC-BY-SA-2, via Commons

Casting

Transforming a type in another, in some “natural” way.

An example is worth 1000 words

- Integer to float: $3 \rightarrow 3.0$
- Integer to string: $3 \rightarrow '3'$
- String to integer: $'3' \rightarrow 3$
- Boolean to integer: $\text{true} \rightarrow 1, \text{false} \rightarrow 0$
- String to boolean: $'foobar' \rightarrow \text{true}, '' \rightarrow \text{false}$

Core idea

- You start with **variables** to get info about the edit
- Variables can be manipulated with **functions**
- You **compare** the value to some “expected” value
- Conditions like these can be **joined** together logically (i.e. “A and B” vs “A or B”)

Variables

Variables is where the whole process begins: they're like buckets, each one has a name and a (pre-filled) value. Depending on your intentions, you'll have to figure out what variables to use. And to do that, you need to know what the (main) variables are.

Variables

Variables can be broken down into four groups:

- About the page being edited
- About the user performing the edit
- About the edit itself
- Generic

Page-related variables

These variables offer generic information about the page. Their names start with “page_”; in the past, it was “article_” (keep this in mind if you're reading old guides).

Page-related variables

- **page_title** - The title of the page, without namespace
“Wikipedia:Five pillars” → “Five pillars”
- **page_namespace** - The (numeric) namespace of the page
“Wikipedia:Five pillars” → 4
- **page_prefixedtitle** - Title with namespace
“Wikipedia:Five pillars” → “Wikipedia:Five pillars”

Page-related variables

You're not going to need these very often, but:

- **page_id** - Unique ID of the page, from action=info
- **page_age** - Number of seconds since page creation
- **page_recent_contributors** - List of usernames
- **page_last_contributor** - Single username

User-related variables

These variables offer generic information about the user performing the edit. Their names start with “user_”.

User-related variables

- **user_name**
- **user_editcount**
- **user_groups** - All user groups, even “automatic” ones
Example: ["*" , "user" , "sysop"]
- **user_rights** - List of rights for those groups
Example: ["read" , "edit" , "createaccount"]

Edit-related variables

These variables offer generic information about the ongoing edit. There's no fixed prefix for these.

There are other variables for actions other than edits, but we're not going to cover those.

Edit-related variables

- **summary** – Note: it doesn't include the auto-generated part
Example: `"/*XYZ*/ new section" → "XYZ"`
- **old_wikitext, new_wikitext**
- **old_size, new_size** – Numeric, in bytes
- **edit_delta** – Integer (can be negative), difference of the above

Edit-related variables

- **added_lines**, **removed_lines** – These are very useful, but very confusing at first. They represent things that are highlighted in the diff view. Each of these is a list of added [removed] lines, but can be thought of as a string of everything that was added [removed].

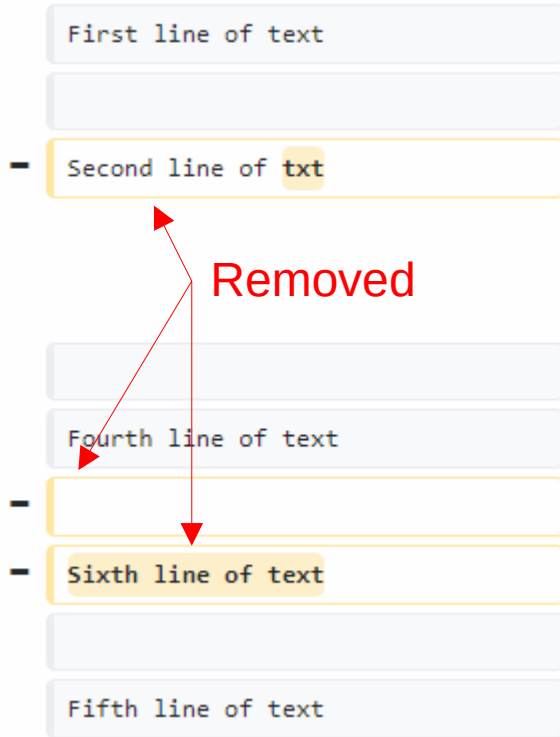
added_lines / removed_lines

Caveats:

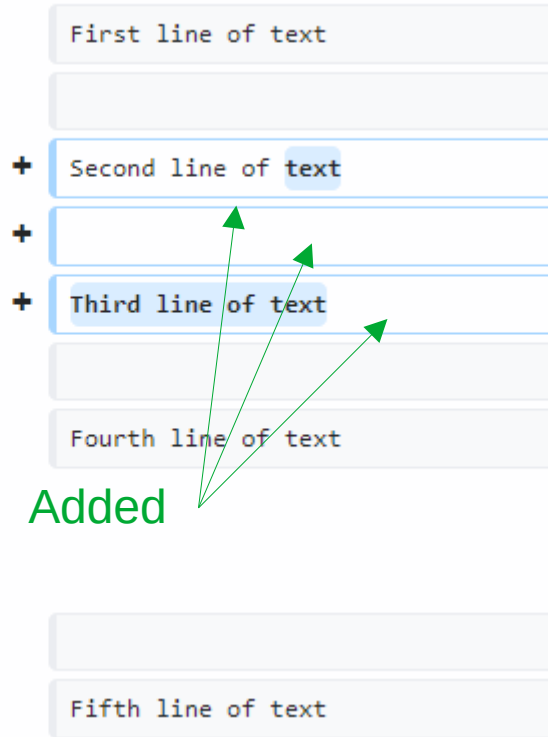
- Not everything in added_lines [removed_] was really added [removed];
- Text in these variables is not transformed (signatures aren't expanded, nor are pipe tricks, subst, etc.)

added_lines / removed_lines

Line 1:



Line 1:



removed_lines:

- "Second line of txt"
- ""
- "Sixth line of text"

added_lines:

- "Second line of text"
- ""
- "Third line of text"

Edit-related variables

- **added_links**, **removed_links** – Same format and similar caveats as `added_/removed_lines` (if you change a link, it will be in both `added_` and `removed_`). Links are taken from the wikitext parser, so it won't be tricked e.g. by “`<nowiki>https://example.org</nowiki>`”.
- **old_links**, **all_links** – Where “all” means “new version only”.

Generic variables

Anything that doesn't fit the previous categories :-)

Generic variables

- **action** – What action is being performed. One of:
'edit', 'move', 'createaccount', 'delete',
'autocreateaccount', 'upload', 'stashupload'
- **timestamp** – UNIX timestamp of the edit, = number of seconds since Jan 1 1970 (you can find converters online)
Example: 15 Jan 2001 00:00 → '979516800'

Functions

Functions let you manipulate “expressions” in various ways. Think of a function as a black box, you put something in (“arguments”), and it spits something else out, depending on what you gave it. Somehow similar to math functions, if you're familiar with calculus.

An “expression” can be a variable, the result of calling another function, etc.

Functions

There's a lot of built-in functions to cover most use cases.

We're only going to examine a selection of commonly-used functions.

Functions

- **lcase (x)**, **ucase (x)** – Convert x to lowercase [upper]
lcase ('DARTH Vader') → 'darth vader'
ucase ('DARTH Vader') → 'DARTH VADER'
- **length (x)** – Length of a string, or number of items in a list
length ('Leia') → 4
length (['Luke', 'Anakin']) → 2

Functions

- **ccnorm(x)** (l33t killer) – Replaces confusable characters in x (and capitalizes it)

```
ccnorm('I h4x0r u n00b') → 'I HAXOR U NOOB'
```

- **norm(x)** – Like ccnorm, but also removes doubles, special characters, and whitespace. Truly a sledgehammer.

```
norm('!ω.ḥk ι.ρρε.Ḃ@1%α!') → 'WIKIPEDIA'
```

```
norm('http://000.op') → 'HTPOOP' (!)
```



Functions

- **count(x, y)** – How many times x is found in y
`count('a', 'I are a lolcat')` → 3
- **rcount(x, y)** – Same as count, but x is a regexp
`rcount('[a-z]', '123baz')` → 3

Spoiler: **r** means
“regexp alert”

Functions

- **contains_any(x, a, b, c, ...)** – Whether x contains any of a, b, c, ...
contains_any('foobar', 'goat', 'cat', 'bar') → true
contains_any('foobar', 'goat', 'cat') → false
- **contains_all(x, a, b, c, ...)** – What is says on the tin
contains_all('foobar', 'cat', 'bar') → false
contains_all('foobar', 'foo', 'bar') → true

Functions

- `equals_to_any(x, a, b, c, ...)` – Whether x is the same of any among a, b, c, ...
 `equals_to_any('foo', 'bar', 'baz', 'foo')` → true
 `equals_to_any('foo', 'bar', 'baz')` → false

Variadic
=
The sky's the limit

Variadic

=

The sky's the limit

PHP process memory



WIKIMEDIA
FOUNDATION

Functions

- **strpos (haystack, needle)**

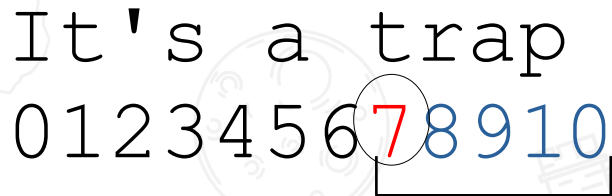


??

Functions

- **strpos (haystack, needle)** – Get the position in which needle is found inside haystack, starting from 0
`strpos('Join the dark side', 'force') → false`
`strpos("It's a trap", 'trap') → 7`

It's a trap
012345678910



Functions

- **substr(str, start, len)** – Extract from str the substring starting at position start and of length len
`substr('Dark side', 5, 4) → 'side'`

Dark side
012345678
 } **side**

Keywords

Think of them as functions...

...but without parentheses.

And they only take two arguments, one on the left of the keyword, and one on the right.

Keywords

- **x in y** – Whether the string X is contained in the string Y
`'bi' in 'Obi-Wan' → true`
`'Luke' in 'Darth Vader' → false`

Essentially equivalent to manipulating the result of
`strpos(Y, X)`
but more readable, isn't it?

Keywords

Caveat: For lists, it doesn't work as you might expect!

```
'Han' in [ 'Han', 'Chewie' ] → true
```

```
'n\nC' in [ 'Han', 'Chewie' ] → true (!)
```

What is happening is: the elements in the list are “glued” together with line breaks (' \n '), so internally what it does is:

```
'n\nC' in 'Han\nChewie' → true
```

Keywords

- **Y contains X** - Same as in, but in reverse order
 - 'Obi-Wan' contains 'bi' → true
 - 'Darth Vader' contains 'Luke' → false

The order is really the only difference, so everything still applies here.

Spoiler!

Keywords

- **x** **rlike** **y** – Whether the string X matches the regular expression Y

'The answer is 42' rlike 'The answer is (not)? \d+' → true

'And the question?' rlike 'The answer is \d+' → false

Caveat: It is case-sensitive

'YODA' rlike 'yoda' → false

Keywords

- **X** **irlike** **Y** - Like rlike, but case-insensitive

'The answer is 42' rlike 'The answer is (not)? \d+' → true

'YODA' irlike 'yoda' → true

"it's a trap" irlike 'a TRAP' → true

Spoiler: **i** means
“case-insensitive”

Keywords

Medical disclaimer: Once you'll be writing filters daily, you'll start seeing `irl`'s everywhere.

And for a good reason: it's darn useful.



I aM vANdaLizIng ThE Wiki

irlIKE 'i am vandalizing the wiki' → true

Avoiding non-free images

sO VerSaTILE, aS I sAId

**Now we have the ingredients.
Enter the mixer:**



Operators

Operators act as glue for expressions.

There are 3 main types of operators, depending on how they glue things together:

- Arithmetic operators (+, -, *, /, %, **)
- Comparison operators (<, <=, >, >=, =, ==, ===, !=, !==)
- Logical operators (!, &, |, ^)

Arithmetic operators

Allow manipulating numeric (integer or float) values. As you would probably expect.

- Addition: +, subtraction: -, multiplication: *, division: /
- Power: ** $\rightarrow 2^{**} 4 = 2 * 2 * 2 * 2 = 16$
- Modulo: % \rightarrow “remainder of Euclidean division”

Remind... of what??

Modulo

How mathematicians
write it

How computer
scientists write it

“ $X \pmod m$ ”, a.k.a. “ $X \% m$ ” means: divide X by m with remainder. The remainder is the result of the modulo.

- $87 \% 14 = ? \rightarrow 87 = 14 * 6 + 3 \rightarrow 87 \% 14 = 3$
- $84 \% 14 = ? \rightarrow 84 = 14 * 6 + 0 \rightarrow 87 \% 14 = 0$

The maximum possible to
have a non-negative remainder

Modulo (properties)

Assuming “ $X \pmod m = Y$ ”:

- Y is always between 0 and $m-1$
- Y is 0 iff m is a divisor of X
- Examples: tell if a number is even, extract hour from timestamp (idea: if you add 24 hours to now, the hour remains the same)
- A lot of beautiful other properties, but they don't fit in this slide



Gauss and Galois be praised



... and Fermat, too



Approved

Comparison operators

Allow testing whether two values are equal/different.

- Inequalities: $<$, $<=$, $>$, $>=$ – What you expect
- Equality: $(=,)$ $==$, $===$, $!=$, $!==$
 - The “!” means “different”, graphic representation of \neq
 - Why keep adding $=$ s? What about $=====$?

Comparison operators

`==` is a “loose comparison”. `===` is a “strict comparison”. The latter checks not only the values, but also the types.

In fact, the former casts its operands to strings before doing the comparison. You should almost never use it.

Comparison operators

- `'Luke' == 'Luke' → true`
- `1 == '1' → true`
- `3.14 == 3.14 → true`
- `true == 1 → true`
- `null == false → true`
- `'Luke' === 'Luke' → true`
- `1 === '1' → false`
- `3.14 === 3.14 → true`
- `true === 1 → false`
- `null === false → false`

Logical operators

What really allows gluing expressions together.

- NOT: $!$
- AND: $\&$
- OR: $|$ (inclusive OR, Latin *vel*)
- XOR: \wedge (eXclusive OR, Latin *aut*)

Logical operators

Truth table

X	Y	$\neg X$	$X \ \& \ Y$	$X \ \ Y$	$X \ \wedge \ Y$
false	false	true	false	false	false
false	true	true	false	true	true
true	false	false	false	true	true
true	true	false	true	true	false

One last word on ambiguity

When creating programming languages (which goes well beyond the scope of this talk), you must pay attention to ambiguity. You might wonder what could be ambiguous here.

Apparently trivial question: what does $1 + 2 + 3$ means?

- Add 1 and 2, then add 3 to the result (LTR)
- Add 3 and 2, then add 1 to the result (RTL)

One last word on ambiguity

Wait. Isn't it the same? Well, yes, but actually no.

In this specific example, it *is* the same just because addition (of [complex] numbers) is an associative (and commutative) operation:

$$(1 + 2) + 3 = 1 + (2 + 3)$$

Are there non-associative operations? That is, without digging into exoteric abstract algebra?

One last word on ambiguity

Yes: what about

```
true | true & false?
```

- `true | true` is `true`. And `true & false` is **false**
- `true & false` is `false`. And `true | false` is **true**

And it gets even worse if you mix up different operand types:

```
true | true === false
```


One last word on ambiguity

So how to resolve this?

People who create programming languages choose a predefined order, that is usually LTR, plus an order in which operators are checked (e.g. comparison first, then logic; think of arithmetic).

What if you want to change the order locally?

Parentheses to the rescue

It works exactly the same as for mathematical operations: you add parentheses to specify the order you want. If you don't use parentheses, the default ordering is used.

- `true | (true & false)` ← Forces RTL
- `(true | true) === false` ← Forces | before ===

Parentheses to the rescue

Pro tip: parentheses usually make the code more readable. Never be afraid to use them, unless the operator precedence is obvious. You also don't need to remember operator precedence, as long as you use parentheses.

Putting it all together

The “tools” presented so far will allow you to formally express a set of conditions about a given edit. This is what we're going to use when writing rules.

Time will teach you how to put things together in the best way possible. Usually, you have multiple ways to do that.

What rules looks like

now that we know how to read them

```
1  !("confirmed" in user_groups) &
2  page_namespace == 0 &
3  (
4    new_size < 50 & old_size > 300 |
5    new_size/(old_size + 1) < 0.1
6  ) &
7  !(lcase(new_wikitext) rlike "(lol)?cats")
8
```

Consequences

Consequences

Each filter has its own enabled consequences. When the rules match an edit, all consequences are activated.

An “automatic” consequence (that cannot be disabled) is logging the filter match in a dedicated log.

What consequences looks like

Actions to take when matched

- Trigger actions only if the user trips a rate limit → Throttle
- Trigger these actions after giving the user a warning → Warn
- Prevent the user from performing the action in question → Disallow
- Revoke the user's autoconfirmed status → (Blockautopromote)
- Block the user and/or IP address from editing → Block
- Tag the edit for further review → Tag

Disallow

Prevents the edit from being saved, showing a custom message to the user.

Prevent the user from performing the action in question

System message to use for disallowing:

abusefilter-disallowed

Page name of other message:

abusefilter-disallowed

(without "MediaWiki:" prefix)

[Show/Hide preview of selected message](#)

[Create/Edit selected message](#)

Disallow





The messages used must be in the MediaWiki namespace, and should be named “MediaWiki:Abusefilter-disallowed-XYZ”, where XYZ can be anything.

You can use custom formatting etc. inside messages.

Disallow

Editing FOOO

Error: UR EDIT ARE NOT SAVD, U NOOB

B *I* |    ▶ [Advanced](#) ▶ [Special characters](#) ▶ [Help](#)  ▼

I NO LAIK LOLCATZ

Warn

Prevents the **first** edit attempt, but allows saving if the user clicks “Save” again.

Trigger these actions after giving the user a warning

System message to use for warning:

abusefilter-warning

Page name of other message:

abusefilter-warning

(without "MediaWiki:" prefix)

Show/Hide preview of selected message

Create/Edit selected message

Warn

Like for “disallow”, the messages used must be in the MediaWiki namespace, but this time they should be named “MediaWiki:Abusefilter-warning-XYZ”, where XYZ can be anything.

You can use custom formatting etc. inside messages.

Warn

The user experience is also identical to “disallow” for the first edit attempt. The second attempt depends on what other actions are enabled for the filter. To put it simply, the result on the second attempt is what you'd get if warn wasn't enabled at all.

Tag

Adds some change tags to the edit, if it can be saved. The user won't get any warning.

Tag the edit for further review

Tags to apply:

mytag1 X
mytag2|

This tag was already selected

Type here and hit Enter to add another tag

Block

Blocks the user with given expiry. Often useful, but use with care! You don't want to abuse this.

Block the user and/or IP address from editing

Block the user and/or IP address from editing
their own talk page

Block duration for anonymous users:

1 settimana



Block duration for registered users:

infinito



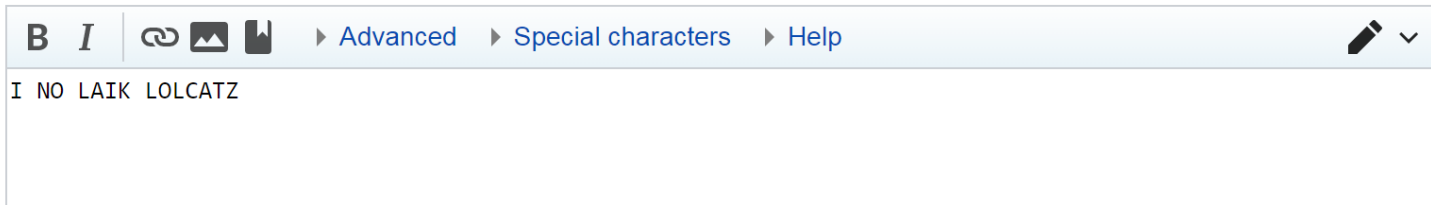
Block

The block will be issued immediately, before saving the edit. It will be logged as performed by a fake user, named e.g. “Abuse filter” (localized).

Block

Editing FOOO

Error: This action has been automatically identified as harmful, and you have been prevented from executing it. In addition, to protect <wiki name>, your user account and all associated IP addresses have been blocked from editing. If this has occurred in error, please contact an administrator. A brief description of the abuse rule which your action matched is: <description>



Throttle

By far the most complicated consequence, but very useful. Using throttle, you “allow” a certain number of edits, until some conditions are met (similar to “warn”). These conditions are preserved between different edits. “Allow” means: any other action of **that filter** is temporarily disabled.

Three parameters: count, period, and groups.

Throttle

Trigger actions only if the user trips a rate limit

Number of actions to allow:

3

Count

Period of time (in seconds):

Period

60

Group throttle by:



user X

Split with commas to join with AND, and insert one by one to join with OR

Groups

Throttle

The options for “groups” are complicated. To explain, let's suppose that you can only specify a single throttle group, “user”.

In reality, there are more options, and they can be combined logically, but it's not hard to understand once you get the basic idea.

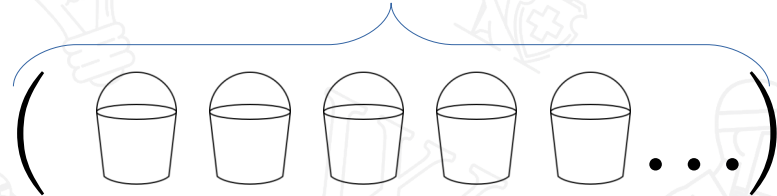
High-quality graphics™

Throttle

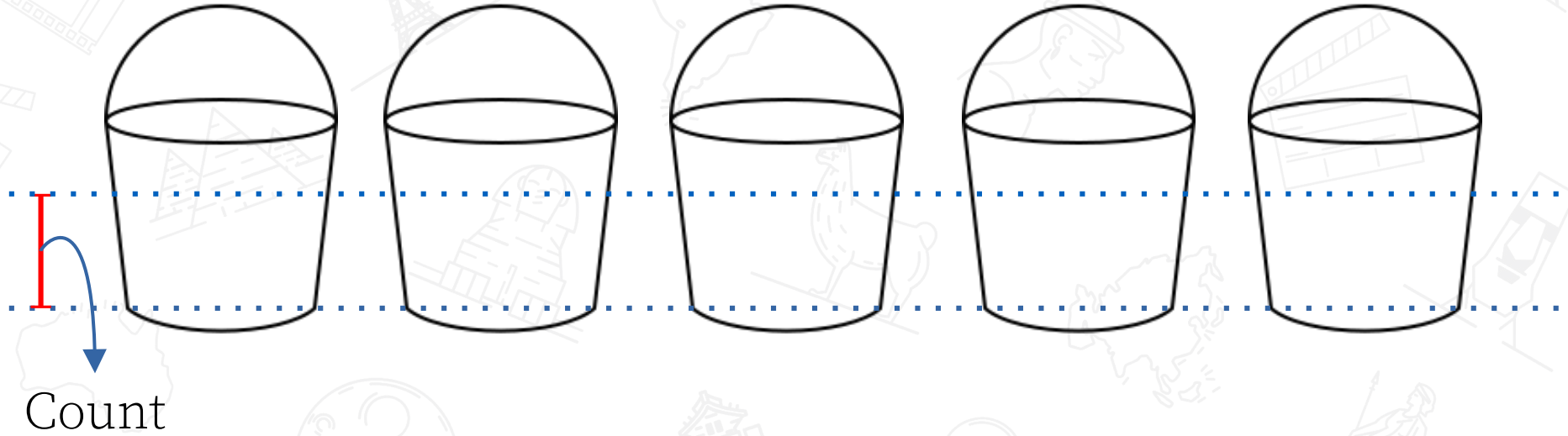


Buckets for filter X

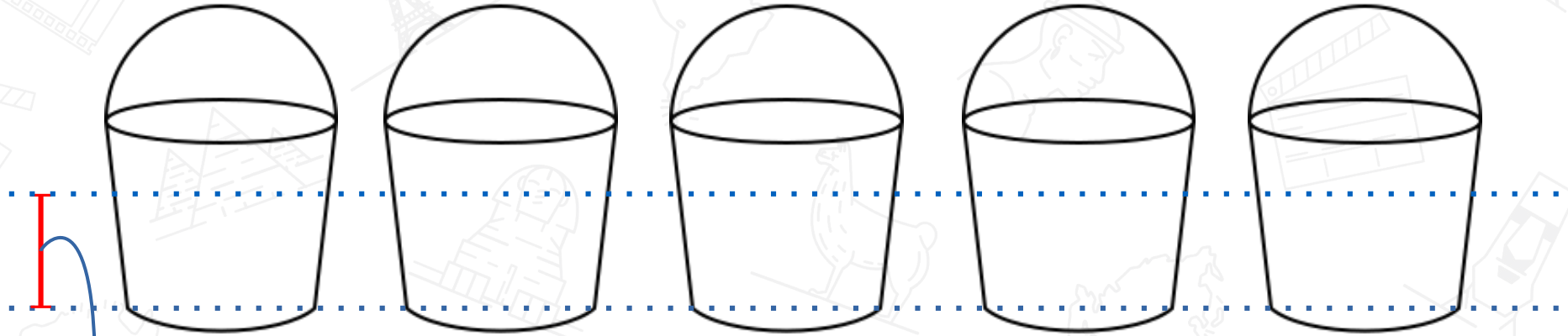
Buckets for other filters



Throttle



Throttle



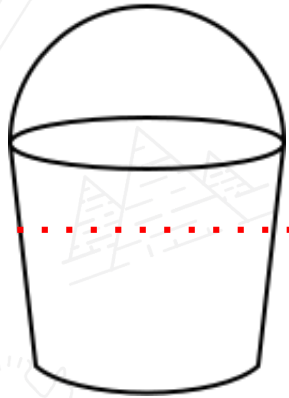
Count



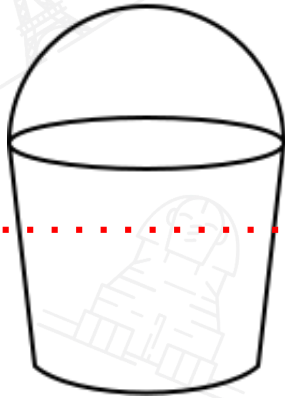
Period



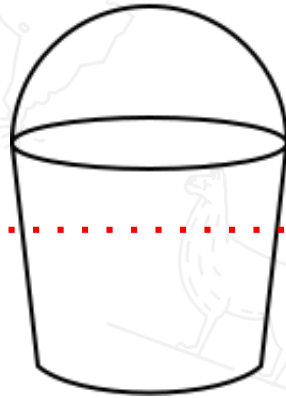
Throttle



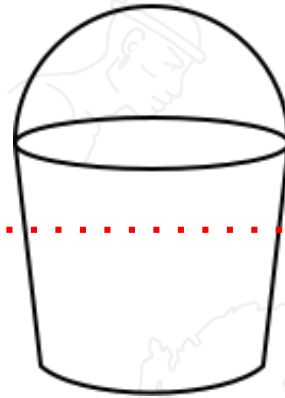
User 1



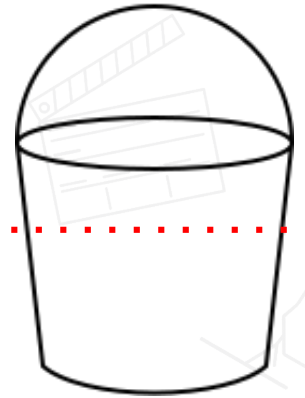
User 2



User 3



User 4

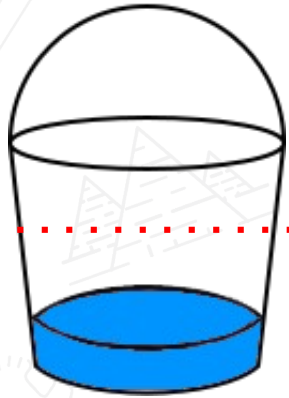


User 5

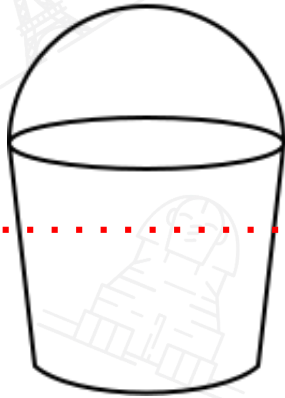


WIKIMEDIA
FOUNDATION

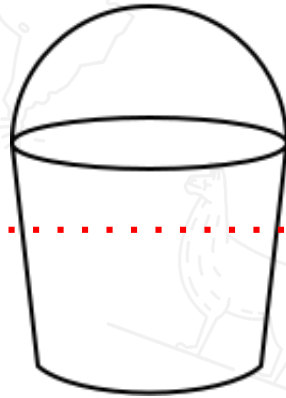
Throttle



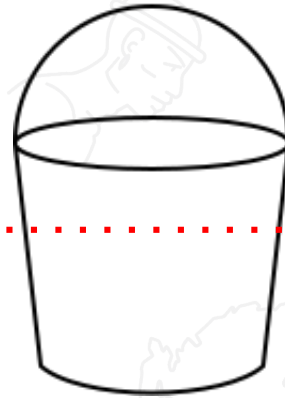
User 1



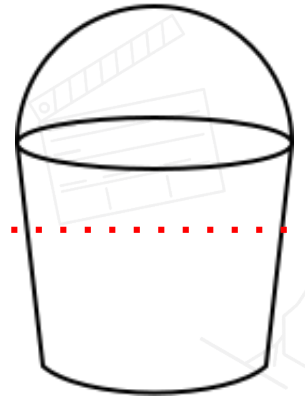
User 2



User 3



User 4

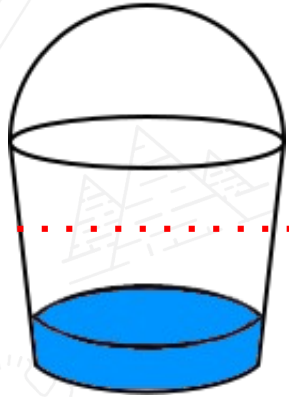


User 5

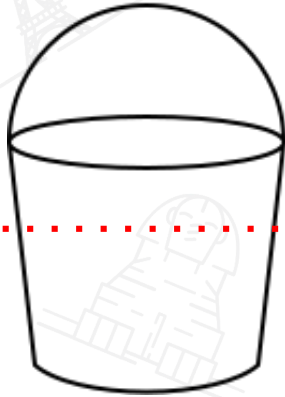


WIKIMEDIA
FOUNDATION

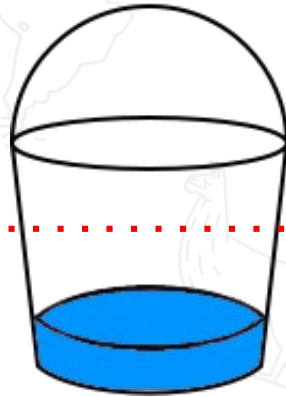
Throttle



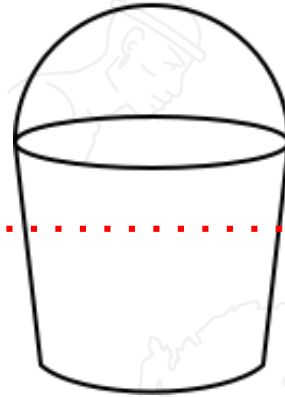
User 1



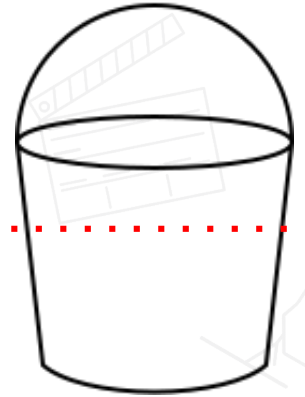
User 2



User 3



User 4

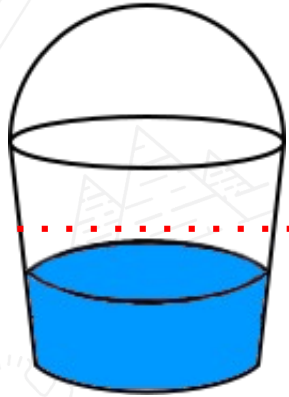


User 5

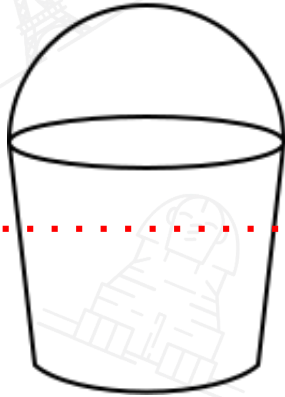


WIKIMEDIA
FOUNDATION

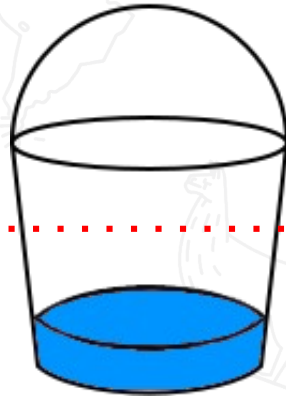
Throttle



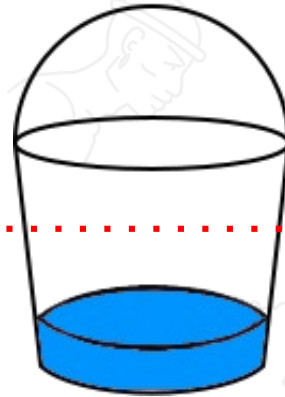
User 1



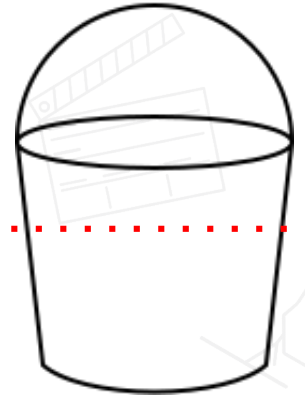
User 2



User 3



User 4

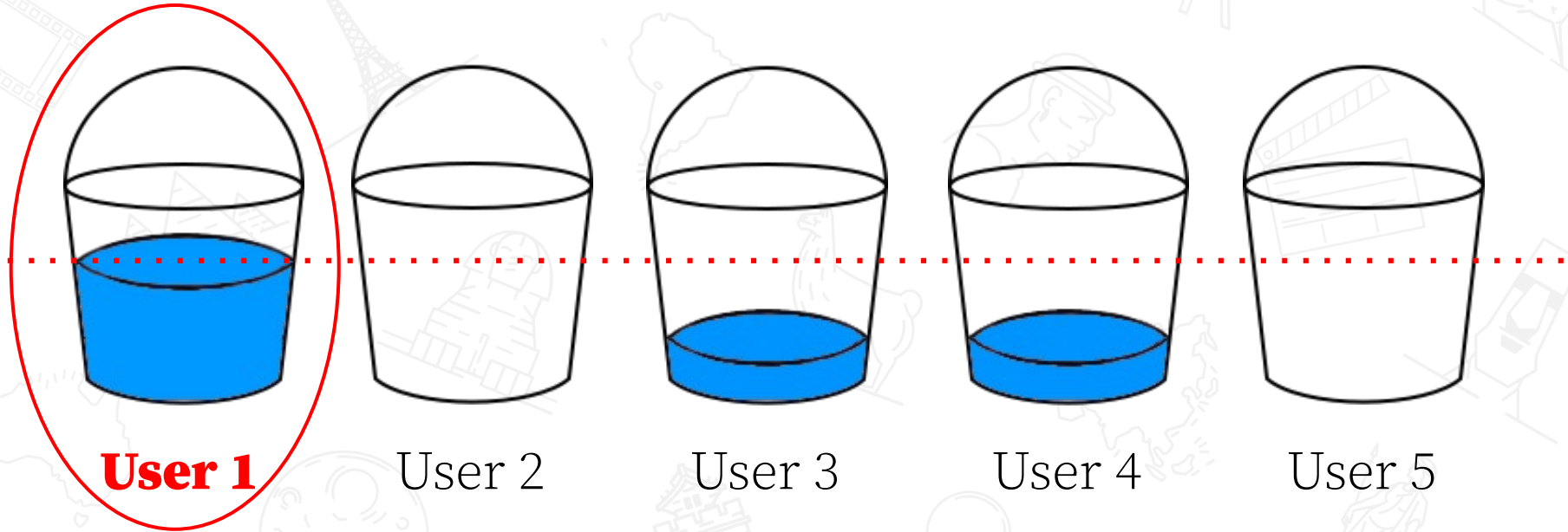


User 5



WIKIMEDIA
FOUNDATION

Throttle



Throttle

The limit was hit. If the user trips this filter another time, every other action of that filter will be activated.

Note that throttle alone without other actions is not useful, because you need to specify what should happen once the limit is reached.

Throttle

Ideas for what's left:

- Add more types of buckets: “page”, “creationdate”, “site”
- Have a single bucket take more than one type: “user” + “page”, so the same user must edit the same page for the level to increase
- Have buckets of different types, but the limit is reached once the first type reaches capacity (regardless of the others)





Now we know things in theory

But what about the practice?

Interface

AbuseFilter provides several special pages. Some are actually subpages of a special page.

- [Special:AbuseFilter](#) (lists active filters)
- [Special:AbuseFilter/42](#) (for editing filter 42)
- [Special:AbuseFilter/test](#) (test a filter against recent changes)
- [Special:AbuseFilter/tools](#) (evaluate a filter rule)
- [Special:AbuseFilter/import](#) (see links at the end of the presentation)
- [Special:AbuseLog](#) (“special” log of all filter hits)

Special:AbuseFilter

List of all active filters, in tabular format.

The list is searchable and sortable (useful there are many filters).

Abuse filter management

Handy navigation links

Abuse Filter navigation (Home | Recent filter changes | Examine past edits | Abuse log | Batch testing | Debugging tools)

Welcome to the Abuse Filter management interface. The Abuse Filter is an automated software mechanism of applying automatic heuristics to all actions. This interface shows a list of defined filters, and allows them to be modified.

Create a new filter Import filter

What it says on the tin

All filters

Options

Deleted filters:

Include deleted filters Hide deleted filters Show only deleted filters

Further options:

Hide private filters Hide disabled filters

Search within rules:

Number per page:

Update

Search options

Filter ID	Public description	Consequences	Status	Last modified	Visibility	Hit count
1	People hating lolcats	Disallow	Enabled	16:14, 11 November 2020 by Daimona Eaytoy (talk contribs block)	Private	0 hits

One filter per row

Special:AbuseFilter/1

Page for editing the filter number 1 (replace with ID of your interest).

Special:AbuseFilter/new is for creating a new filter (same interface).

Filter ID: 1

Filter ID: unique (and reliable) identifier

Description:

People hating lolcats

(publicly viewable)

Filter name: user-friendly identifier

Filter hits: 1 hit

Statistics: Of the last 4 actions, this filter has matched 1 (25%). On average, its run time is 1.25 ms, and it consumes 2 conditions of the condition limit.

```

1 | ("user" in user_groups) &
2 | lcase(added_lines) contains "i hate lolcats" &
3 | !( lcase(removed_lines) contains "i hate lolcats" )
4

```

Rules - the heart of the filter

Conditions:

Select an option to add it at the cursor

Switch editor

Check syntax

Check rules validity • • •

Old-style editor, plain text box. Useful for certain characters, or mobile web interface.



Loicats will take over the world. --CatLover 15 January 2008

Notes

Notes - to explain the intentions of the filter, what changes you made, link to examples, etc.

Hide details of this filter from public view

Flags: Enable this filter

Mark as deleted

Filter flags. “Hidden” is vital for many kinds of filters

Filter last modified: 19:36, 11 November 2020 by Daimona Eaytoy ([talk](#) | [contribs](#) | [block](#))

History: [View this filter's history](#)

[Revert actions taken by this filter](#)

Tools: [Test this filter against recent edits](#)

[Export this filter to another wiki](#)





Actions to take when matched

- Trigger actions only if the user trips a rate limit
- Trigger these actions after giving the user a warning
- Prevent the user from performing the action in question

System message to use for disallowing:

abusefilter-disallowed



Page name of other message:

abusefilter-disallowed

(without "MediaWiki:" prefix)

Show/Hide preview of selected message

Create/Edit selected message

- Revoke the user's autoconfirmed status
- Block the user and/or IP address from editing
- Block the respective IP range from which the user originates
- Remove the user from all privileged groups
- Tag the edit for further review

Actions (already covered before)

Save filter

Don't forget to save!

Special:AbuseFilter/test

Test a rule against 100 recent edits.

Always use it before creating a new filter, or modifying an existing one!

Straightforward interface, 90% of the times it's just:

- Enter rules
- Hit “test”

Special:AbuseLog

List of every filter hit. “Special” log, not entirely publicly viewable.

Lots of search options, by filter ID, by action taken, by target user/page, etc.

Special:AbuseLog

Search options (click to expand)

▼ Search the abuse log

This log shows a list of all actions caught by the filters.

(newest | oldest) View (newer 3 | older 3) (20 | 50 | 100 | 250 | 500)

- | | Filter IDs | Target page | Filter names |
|---|----------------------------|---------------------------|--|
| • 00:43, 1 December 2020: Daimona Eaytoy (talk contribs block) triggered filter 267 performing the action "edit" on Main page . Actions taken: none; Filter description: No actions here (details examine) | filter 267 | Main page | No actions here (details examine) |
| • 00:43, 1 December 2020: Daimona Eaytoy (talk contribs block) triggered filter 266 performing the action "edit" on Main page . Actions taken: Block; Filter description: Test msg (details examine) | filter 266 | Main page | Test msg (details examine) |
| • 00:42, 1 December 2020: Daimona Eaytoy (talk contribs block) triggered filter 267 performing the action "edit" on Main page . Actions taken: none; Filter description: No actions here (details examine diff) | filter 267 | Main page | No actions here (details examine diff) |

(newest | oldest) View (newer 3 | older 3) (20 | 50 | 100 | 250 | 500)

Handy links

Tricks of the trade

Pitfalls

Some common mistakes that even the most experienced filter editors might make.

For a good list, see <https://w.wiki/oqs> (help page on enwiki).

Pitfalls

One very common mistake is to use the `edit_delta` variable without checking if the action is an edit. For if the action is not an edit, `edit_delta` is null, and

```
edit_delta < -100000
```

is true, because `null` is “smaller” than every number.

Performance

Usually you should not worry about performance (see WP:PERF on enwiki), but you should know that some variables require quite a lot of time to be computed.

Fancy graphs are available at <https://w.wiki/oqz>

Performance

Some examples of slow variables are `page_recent_contributors` and variables that contain the links on the page (e.g. `added_links`). Think twice before using them, but don't be scared, and do use them if you need to.

Performance

Another optimization is taking advantage of “short-circuiting” by putting first the conditions that are less likely to be true (with the AND operator), or more likely to be true (with the OR operator).

There's no time to discuss it, but keep this in mind if you read these slides again after having learned some more syntax.

Tips and tricks

- Always check `user_groups`. Most filters are meant to address vandalism or spam, which usually comes from non-confirmed users. Check this with
! ("confirmed" in `user_groups`)

This is also necessary to avoid false positives, i.e. blocking an experienced user by mistake.

Tips and tricks

- To check whether some text was added, always check `added_lines` AND `removed_lines`, to ensure that it wasn't already present (remember that `added_lines` is not a faithful representation of what was added). To see if “vader” was added:

```
added_lines contains "vader" &  
!(removed_lines contains "vader")
```

Tips and tricks

- When you create a new filter, do not enable any action. Let it run for a few days, monitor which edits it matches, and tweak it if necessary. Enable “destructive” options (e.g. block, disallow) only when you're confident that the filter will have close to 0 false positives.

Limitations

AbuseFilter is powerful, but it can't do everything. Notable examples of things it can't do while filtering an edit:

- Get information about a past edit
- Get any information that is not stored in a built-in variable
- Completely replace manual intervention!
- But if you think a feature is missing, you can file a task on Phabricator

Examples

Where can I test a filter?

Testing a filter for real would require using a real wiki, and then checking the filter against recent changes (using /test).

We won't be doing this because:

- We'd need to set up privileged accounts on a wiki
- We'd need the recent changes of that wiki to match the filters that we want to test

Where can I test a filter?

This would be too complicated, so take it as a mental exercise for being able to understand a filter without trying it.

Most of the examples use syntax highlighting so it's easier to recognize variables, functions etc.

Example 0

Fool me once: what does the following filter do?

```
! ("confirmed" in user_groups &  
page_namespace === 0
```

Example 0

Fool me once: what does the following filter do?

```
! ("confirmed" in user_groups) &  
page_namespace === 0
```

Nothing, there's a missing parentheses :-)

Always use that "check syntax" button. Some logic error won't be detected, thus double-checking is always a good idea.

Example 1

Rewind: what does the following filter do? (Now with missing ")")

```
! ("confirmed" in user_groups) &  
page_namespace === 0
```

Example 1

Rewind: what does the following filter do? (Now with missing ")")

```
!("confirmed" in user_groups) &  
page_namespace === 0
```

Answer: it finds edit by non-confirmed users on the mainspace.

Example 2

Play it again, Sam: what does the following filter do?

```
1  !("confirmed" in user_groups) &
2  page_namespace == 0 &
3  (
4    new_size < 50 & old_size > 300 |
5    new_size/(old_size + 1) < 0.1
6  ) &
7  !(lcase(new_wikitext) rlike "(lol)?cats")
8
```

Example 2

Play it again, Sam: what does the following filter do?

```
1  !("confirmed" in user_groups) &
2  page_namespace == 0 &
3  (
4    new_size < 50 & old_size > 300 |
5    new_size/(old_size + 1) < 0.1
6  ) &
7  !(lcase(new_wikitext) rlike "(lol)?cats")
8
```

- User is not confirmed;
- Page is in the mainspace;
- Page was big and is now small, OR ratio of removed content is high;
- The new content of the page doesn't contain "cats" nor "lolcats".

Example 3

Against all odds: what does the following filter do?

```
1 user_editcount < 12 &
2 new_size < 100 &
3 page_namespace % 2 == 1 &
4 old_size > 150 &
5 !(user_name in page_prefixedititle) &
6 page_namespace != 3 &
7 !(added_lines irlike "^#\s*redirect\s*\[\[")
8
```

Example 3

Against all odds: what does the following filter do?

```
1 user_editcount < 12 &
2 new_size < 100 &
3 page_namespace % 2 == 1 &
4 old_size > 150 &
5 !(user_name in page_prefixedititle) &
6 page_namespace != 3 &
7 !(added_lines irlike "^#\s*redirect\s*\[\[")
8
```

- User made less than 12 edits;
- Page is now short;
- Page is a talk (odd namespace);
- Page was not short;
- Page title doesn't contain the current user's name;
- Page is not a user talk;
- Page wasn't made a redirect;

In short, new users blanking non-user talk pages.

Example 4

Our time is running out: what does the following filter do?

```
1  !(user_groups contains "sysop") &
2  equals_to_any( page_namespace, 0, 2, 3, 10 ) & (
3    (
4      ccnorm(added_lines) contains "POOP" &
5      !(ccnorm(removed_lines) contains "POOP")
6    ) |
7    length( added_links ) > 2
8  ) &
9  edit_delta < -1000 &
10 timestamp % ( 60 * 60 * 24 ) ) / 3600 < 7
11
```

Example 4

Our time is running out: what does the following filter do?

```
1  !(user_groups contains "sysop") &
2  equals_to_any( page_namespace, 0, 2, 3, 10 ) & (
3    (
4      ccnorm(added_lines) contains "POOP" &
5      !(ccnorm(removed_lines) contains "POOP")
6    ) |
7    length( added_links ) > 2
8  ) &
9  edit_delta < -1000 &
10 timestamp % ( 60 * 60 * 24 ) ) / 3600 < 7
11 |
```

- User is not a sysop;
- Namespace is 0, 2, 3, or 10;
- (At least) one of:
 - “poop” was added (normalizes confusable characters);
 - 3 or more links were “added”;
 - 1000 or more bytes were removed
- Earlier than 7 a.m. UTC

Useful guides

- <https://w.wiki/oy7> ([[[:mw:Extension:AbuseFilter/Rules format]]]), a complete guide with all variables, functions, examples, tips, etc.
- <https://w.wiki/oy8> ([[[:mw:Extension:AbuseFilter/Actions]]]), describes actions, especially “throttle”.
- <https://w.wiki/oyA> ([[[:meta:Small wiki toolkits/Starter kit/AbuseFilter]]]), quick tutorial

THANK YOU
And may the force be with you

