regex
regex

Description

Removes results that do not match the specified regular expression.

Syntax

The required syntax is in **bold**.

```
regex
    (<field>=<regex-expression> | <field>!=<regex-expression> | <regex-expression>)
```

**Required arguments**

`<regex-expression>`

**Syntax:** `"<string>"`

**Description:** An unanchored regular expression. The regular expression must be a Perl Compatible Regular Expression supported by the PCRE library. Quotation marks are required.

**Optional arguments**

`<field>`

**Syntax:** `<field>`

**Description:** Specify the field name from which to match the values against the regular expression. You can specify that the `regex` command keeps results that match the expression by using `<field>=<regex-expression>`. To keep results that do not match, specify `<field>!=<regex-expression>`. **Default:** `_raw`

Usage

The `regex` command is a distributable streaming command. See Command types.

Use the `regex` command to remove results that do not match the specified regular expression.

Use the `rex` command to either extract fields using regular expression named groups, or replace or substitute characters in a field using `sed` expressions.

When you use regular expressions in searches, you need to be aware of how characters such as pipe ( | ) and backslash ( \ ) are handled. See SPL and regular expressions in the Search Manual.

For general information about regular expressions, see About Splunk regular expressions in the Knowledge Manager Manual.

Examples

**Example 1:** Keep only search results whose "_raw" field contains IP addresses in the non-routable class A (10.0.0.0/8). This example uses a negative lookbehind assertion at the beginning of the expression.

```
... | regex _raw="(?<!\d)10\.|\d{1,3}\.|\d{1,3}\.|\d{1,3}(?!\d)"*
```
Example 2: Keep only the results that match a valid email address. For example, buttercup@example.com.

```regex
^[a-z0-9_.-]+@[\da-z\.-]+\.[a-z\.]{2,6}$
```

The following table explains each part of the expression.

<table>
<thead>
<tr>
<th>Part of the expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>^</code></td>
<td>Specifies the beginning of the string.</td>
</tr>
<tr>
<td><code>([a-z0-9_.-]+)</code></td>
<td>This is the first group in the expression. Specifies to match one or more lowercase letters, numbers, underscores, dots, or hyphens. The backslash (<code>\</code>) character is used to escape the dot (<code>.</code>) character. The plus (<code>+</code>) sign specifies to match from 1 to unlimited characters in this group. In this example this part of the expression matches <code>buttercup</code> in the email address <code>buttercup@example.com</code>.</td>
</tr>
<tr>
<td><code>@</code></td>
<td>Matches the <code>at</code> symbol.</td>
</tr>
<tr>
<td><code>([\da-z\.-]+)</code></td>
<td>This is the second group in the expression. Specifies to match the domain name, which can be one or more lowercase letters, numbers, underscores, dots, or hyphens. This is followed by another escaped dot character. The plus (<code>+</code>) sign specifies to match from 1 to unlimited characters in this group. In this example this part of the expression matches <code>example</code> in the email address <code>buttercup@example.com</code>.</td>
</tr>
<tr>
<td><code>([a-z\.]{2,6})</code></td>
<td>This is the third group. Specifies to match the top-level domain (TLD), which can be 2 to 6 letters or dots. This group matches all types of TLDs, such as <code>.co.uk</code>, <code>.edu</code>, or <code>.asia</code>. In this example it matches <code>.com</code> in the email address <code>buttercup@example.com</code>.</td>
</tr>
<tr>
<td><code>$</code></td>
<td>Specifies the end of the string.</td>
</tr>
</tbody>
</table>

See also

Commands

- `rex`
- `search`