Regex in Your SPL
An Easy Introduction

Michael Simko | Sr. Engineer, Instructor

September 2017 | Washington, DC
Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward looking statements we may make. In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

Splunk, Splunk>, Listen to Your Data, The Engine for Machine Data, Splunk Cloud, Splunk Light and SPL are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. © 2017 Splunk Inc. All rights reserved.
Basics of Regular Expressions

What is this Regex thing all about?
1. Filtering. Eliminate unwanted data in your searches
2. Matching. Advanced pattern matching to find the results you need
3. Field Extraction on-the-fly
### What Is Regex?

<table>
<thead>
<tr>
<th>Source</th>
<th>Quote</th>
</tr>
</thead>
<tbody>
<tr>
<td>w3schools.com</td>
<td>&quot;A regular expression is an object that describes a pattern of characters. Regular expressions are used to perform pattern-matching and ‘search-and-replace’ functions on text.”</td>
</tr>
<tr>
<td>Mastering Regular Expressions, O’Rielly, Jeffery E.F. Friedl</td>
<td>&quot;Regular expressions are an extremely powerful tool for manipulating text and data… <em>If you don't use regular expressions yet, you will...</em>&quot;</td>
</tr>
<tr>
<td>Regexbuddy.com (and others – Original source unknown)</td>
<td>&quot;A regular expression is a special text string for describing a search pattern. You can think of regular expressions as <em>wildcards on steroids.</em>&quot;</td>
</tr>
</tbody>
</table>
Regex Basics
The Main Elements

Control Characters:
^ Start of a Line
$ End of a Line

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character (letter, #, or _)
\W Not a Word Character

Operators:
* Zero or More
+ One or More
? Zero or One

These elements work together to specify a pattern
Regex Basics
The Main Elements

Control Characters:
\^ Start of a Line
\$ End of a Line

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Operators:
* Zero or More
+ One or More
? Zero or One

Sample Regex: \^\d+\s\w+\d+\s\d+:\d+\s\d+:\d+

- is the literal character colon
- without a + or * is a single space
- \w+ is one or more word characters
- \d+ is one or more digits
- ^ Regex is Anchored to the beginning of the line
Regex Basics
The Main Elements

Control Characters:
\^ Start of a Line
\$ End of a Line

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Operators:
* Zero or More
+ One or More
? Zero or One

Sample Regex: \^\d+\s\w+\d+\s\d+:\d+:\d+

Matching String: 22 Aug 2017 18:45:20 On this date, Michael made BBQ references
Regex Basics
To Protect and Give Options

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Protection Characters:
To escape or protect special characters: \ The Backslash or back-whack

Protect periods, [],(),{}, etc when you want to use the literal character

Special Characters:
To give multiple options: 
The pipe character (also called “or”)
Regex Basics
To Protect and Give Options

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Regex: Indiana|Purdue

Purdue 8w 3l .727 19w 5l .792
Indiana 5w 4l .500 15w 8l .652

Regex: \d+.\d+.\d+.\d+

Login Failure From 192.168.12.145
Login Success From 10.35.36.37

(we’ll do the above a different way later)
Regex Basics
Only Some May Pass

Control Characters:
\^ Start of a Line
\$ End of a Line

Special Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[ ] Include
[^] Exclude

Include Characters:
[...]
Example usage: [a-zA-Z0-9]

Exclude Characters:
[^...]
Example usage: [^ ]
Regex Basics
Only Some May Pass

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[ ] Include
[^] Exclude

Special Characters:
| Alternative / “or”

Control Characters:
^ Start of a Line
$ End of a Line

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[ ] Include
[^] Exclude

Regex: server:[a-z0-9]+  Regex: server:[^ ]

Keep going so long as you hit characters that are lowercase a-Z or 0-9

server:253fsf2,host=23423
server: 253fsf2,host=23423
server:253f sf2,host=23423

Go until you hit a space
**Regex Basics**

**Say What Again**

**Control Characters:**
- ^ Start of a Line
- $ End of a Line

**Special Characters:**
- | Alternative / “or”

**Character Types:**
- \s White Space
- \S Not white space
- \d Digit
- \D Not Digit
- \w Word Character
- \W Not Word Characters

**Protection Characters:**
- \ The next character is a literal

**Inclusion Characters:**
- [] Include
- [^] Exclude

**Repetition:**
- {} Number of Repetitions
- {#, #} Range of Repetitions

Repetition is used to define the exact number of characters
Or an upper and lower boundary of acceptable characters
(or the exact number of repetitions of a pattern)
Regex Basics
Say What Again

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Repetition:
{#} Number of Repetitions
{#,#} Range of Repetitions

Protection Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D NotDigit
\w Word Character
\W NotWord Characters

Only 1 line matched because IP format allows 1-3 digits per octet

| Regex: IP: \d{3}\.|\d{3}\.|\d{3}\.|\d{3}
| IP: 172.106.190.100
| IP: 10.24.255.2
| IP: 224.252.2.52

| Regex: IP: \d{1,3}\.|\d{1,3}\.|\d{1,3}\.|\d{1,3}
| IP: 172.16.19.1
| IP: 10.24.255.2
| IP: 224.252.2.52

All 3 lines matched since we account for the IP Address format
Regex Basics
To Protect and Give Options

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Inclusion Characters:
[] Include
[^] Exclude

Protection Characters:
\ The next character is a literal

Logical Groupings:
() Wrap sets of the Regex

Repetition:
{#} Number of Repetitions
{#, #} Range of Repetitions

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Later we’ll use these as “capture groups”

Use to specify repetition for adjacent elements in order to form patterns
Regex Basics
To Protect and Give Options

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Repetition:
{#} Number of Repetitions
{#, #} Range of Repetitions

Logical Groupings:
() Wrap sets of the Regex

Revisiting the IP Matching from a couple of slides ago
Alternate Regex:  IP: (\d{1,3}\.)\{3\}\d{1,3}

IP: 172.16.19.1
IP: 10.24.255.2
IP: 224.252.2.52

Repeats \d{1,3}\. three times
Then tacks on the last \d{1,3}
Regex Basics

The Last (Not so Basic) Element

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Logical Groupings:
() Wrap sets of the Regex

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Repetition:
{#} Number of Repetitions
{#, #} Range of Repetitions

Named Capture Groups:
(?<CaptureGroupName>stuff)

This names the capture group (e.g., logical grouping).
Now when you return the capture, it has a name and not just “Capture Group 1”
Regex Basics

The Last (Not so Basic) Element

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Logical Groupings:
() Wrap sets of the Regex

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Repetition:
{#} Number of Repetitions
{#,#} Range of Repetitions

Named Capture Groups:
(?<CaptureGroupName>stuff)

Regex: `user:\s(?<username>[^@]+)`

Log 1: blah blah user: msimko@splunk.com
Log 2: more blah user: michael@kinneygroup.com

Go until we hit an @
Capture as field username
Anchor off user:\s
Regex in SPL

Using Regular Expressions to improve your SPL
**Fields are fundamental to Splunk Search**

- `erex`
- `rex`
- Interactive Field Extractor
- Props – Extract
- Transforms - Report

**Regex in Your SPL**

**Search Time Regex**

- **Field Extractions**
  - `erex`
  - `rex`
  - Interactive Field Extractor
  - Props – Extract
  - Transforms - Report

- **Evaluation**
  - `Regex`
  - `match`
  - `replace`

**Regex provides granularity when evaluating data**
Field Extractions
On the fly (No need to work ahead)
**erex Command**

**Field Extractions Using Examples**

Use Splunk to generate regular expressions by providing a list of values from the data.

- Scenario: Extract the first word of each sample phrase from `| windbag`
  - Step 1, find the samples
  - Step 2, extract the field
Erex Command: ...

```
| windbag | erex firstwords examples="Une, يؤلمن, Կրնամ" |
```

Easter egg that creates sample data

New Field to create

Examples from the data
erex Command
Field Extractions Using Examples

| windbag | erex firstwords examples="Үңә, ﯽؤلمەن, ﻪپنەٰپ" |

New Field created
The values erex generated based on the samples
erex Command
Field Extractions Using Examples

- Erex is a great introduction to using regular expressions for field extraction.
  - Erex provides the rex that it generated
  - Going forward, use the rex in your saved searches and dashboards.
  - Rex is more efficient
rex Command

Extract Fields Using Regular Expressions at Search Time

Creates a Field Extraction

... | rex field={what_field} "FrontAnchor(?><extraction>{characters}+)BackAnchor"
rex Command

Extract Fields Using Regular Expressions at Search Time

```bash
| windbag | rex field=sample "^(?<FirstWord>[S+]*")"
```

Specify the field to rex from

Front Anchor

Named Field Extraction

Grab any non-space character
rex Command

Extract Fields Using Regular Expressions at Search Time

```
| windbag | rex field=sample "^(?<FirstWord>[S+]*)"
```

Named Field Extraction

Grab any non-space character
rex Command

Use Rex to Perform SED Style Substitutions

SED is a stream editor. It can be used to create substitutions in data.

Splunk uses the rex command to perform Search-Time substitutions.
rex Command

Use Rex to Perform SED Style Substitutions

```
| windbag | search lang="*Norse"
| rex mode=SED "s/\(Norse\)/Not-so-old \1/g"
```

- Set the mode
- s for substitute
- g for global (more than once)
- () to create a capture group
- \1 to paste capture group
- Substitute the stuff between the first / and second / with the stuff between second / and third /
rex Command

Use Rex to Perform SED Style Substitutions

| rex mode=sed "s/Old (Norse)/Not-so-old 1/g" |

Result:

…

| rex mode=sed "s/Old (Norse)/Not-so-old \1/g" |
Evaluation
Using Regular Expressions for Pattern Matching
Regex Command
Filter Using Regular Expressions

sourcetype=fs-notification | regex chgs="^modtime"
Match Function
Filter Using Regular Expressions

match(SUBJECT),"REGEX"

... | eval n = if(match(field,"^MyRegex", 1, 2)
sourcetype=access_combined_wcookie
| eval com = if(match(referer,"http:.*\.com"),"True","False")

Match. Returns 1 for it matches, 0 for not.
Field to evaluate
The Regex
Replace Command
Switch Data at Search Time

Replace field values with the values you specify

... | replace "<whoever>" WITH "<whomever>" IN <target_field>
Replace Command
Switch Data at Search Time

Replace field values with the values you specify

... | replace "<whoever>" WITH "<whomever>" IN <target_field>

| windbag | replace "Euro" with "Euro: How is a currency a language" in lang

String to be replaced

operator

String to replace with

operator

Field in which to make the replacement
Persistence

Regular Expressions That Exist Outside Your Search

Until this point, every one of our extractions have only existed in the search. But, what if we want them to persist? Or to share them?

1. Interactive Field Extractor

2. Extractions in Props / Transforms
Persistent Field Extractions

Comparing The Persistent Field Extractions

Interactive Field Extractor

- Walk-through UI
- You may want to rewrite the generated Regex
- Does not require admin rights

Extract in Props

- Straight editing in props.conf
- Requires Admin Rights (or an admin to put in place)

Report in Transforms

- Edit directly in transforms.conf
- Invoked by props.conf
- Requires Admin Rights (or an admin to put in place)
1. Use Regex to create powerful filters in your SPL
2. Use Regex to create field extractions
3. Regex doesn’t have to be hard. You can do this!
Thank You

Don't forget to rate this session in the .conf2017 mobile app

dsplunk> .conf2017
Appendix A

Caveats
| windbag | search lang="*Norse"
| rex mode=sed "s/\w+\ (Norse)/\w+\ -so-old /g"

Caveat:

The substitution from rex comes after the lang field is extracted. So even though the event data is showing us the substitution, the field lang is showing the original value.
Appendix B

Exercises to Practice With
Regex Basics

The Main Elements

Control Characters:
^ Start of a Line
$ End of a Line

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Operators:
* Zero or More
+ One or More
? Zero or One

Scenario Regex: ^\d+\s\w+\d+\s\d+:\d+:\d+

Learn by Fire:
Which of these will the sample Regex match?

A. 002421 Februari 1083 1:242525:22352
B. 07 Feb 17 12:53:36AM
C. Feb 13 2017 18:46:56
D. 14 February 2017 07:45:47Z

(answers on next slide)
# Regex Basics

## The Main Elements

### Control Characters:

- `^` Start of a Line
- `$` End of a Line

### Character Types:

- `\s` White Space
- `\S` Not white space
- `\d` Digit
- `\D` Not Digit
- `\w` Word Character
- `\W` Not Word Characters

### Operators:

- `*` Zero or More
- `+` One or More
- `?` Zero or One

---

**Scenario Regex:**

```
^\d+\s\w+\d+\s\d+:\d+:\d+
```

---

**Learn by Fire:**

Which of these will the sample Regex match?

A. 002421 Februari 1083 1:242525:22352
B. 07 Feb 17 12:53:36AM
C. Feb 13 2017 18:46:56
D. 14 February 2017 07:45:47:46

Regex doesn’t care if it looks wrong. It only cares if it matches the pattern.
### Regex Basics

#### The Main Elements

<table>
<thead>
<tr>
<th>Control Characters:</th>
<th>Character Types:</th>
<th>Operators:</th>
</tr>
</thead>
<tbody>
<tr>
<td>^  Start of a Line</td>
<td>\s  White Space</td>
<td>*  Zero or More</td>
</tr>
<tr>
<td>$  End of a Line</td>
<td>\S  Not white space</td>
<td>+  One or More</td>
</tr>
<tr>
<td></td>
<td>\d  Digit</td>
<td>?  Zero or One</td>
</tr>
<tr>
<td></td>
<td>\D  Not Digit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\w  Word Character</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\W  Not Word Characters</td>
<td></td>
</tr>
</tbody>
</table>

**Practice:** Create a Regex that describes all three of the following strings

06 February 2017 192.168.1.2
05 Apr 2014 10.2.1.150
31 July 2020 19..15.63
# Regex Basics

## The Main Elements

<table>
<thead>
<tr>
<th>Control Characters:</th>
<th>Character Types:</th>
</tr>
</thead>
<tbody>
<tr>
<td>^ Start of a Line</td>
<td>\s White Space</td>
</tr>
<tr>
<td>$ End of a Line</td>
<td>\S Not white space</td>
</tr>
<tr>
<td></td>
<td>\d Digit</td>
</tr>
<tr>
<td></td>
<td>\D Not Digit</td>
</tr>
<tr>
<td></td>
<td>\w Word Character</td>
</tr>
<tr>
<td></td>
<td>\W Not Word Characters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operators:</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Zero or More</td>
</tr>
<tr>
<td>+ One or More</td>
</tr>
<tr>
<td>? Zero or One</td>
</tr>
</tbody>
</table>

**Scenario:** Create a Regex that describes the following strings

A solution:

```
\d+\s\w+\s\d+\s\d*\.\d*\.\d*\.\d*
```

- 06 February 2017 192.168.1.2
- 05 Apr 2014 10.2.1.150
- 31 July 2020 19..15.63
Regex Basics

The Main Elements

1. Open up your Splunk
2. | windbag | head 20 | table _raw
3. Copy the _raw data
4. Paste the data in Regex101.com

Goals: Extract the following fields for each event:
- lang
- sample
- The Date without Time
- The Time

Perform these as “named” extractions
Replace Command

Switch Data at Search Time

Silly version to try on your own

| windbag | head 20 | replace "1" WITH "Uno" in odd

Try it, then click the down chevron to see the results