# Table of Contents

## Introduction
- Welcome to the Search Reference............................................................. 1
- Understanding SPL syntax.......................................................................... 2
- How to use this manual............................................................................... 7

## Quick Reference
- Splunk Quick Reference Guide................................................................. 10
- Command quick reference........................................................................ 10
- Commands by category............................................................................ 16
- Command types........................................................................................ 28
- Splunk SPL for SQL users........................................................................ 34
- SPL data types and clauses...................................................................... 40

## Evaluation Functions
- Evaluation functions................................................................................ 47
- Comparison and Conditional functions..................................................... 66
- Conversion functions................................................................................. 77
- Cryptographic functions........................................................................... 86
- Date and Time functions.......................................................................... 88
- Informational functions............................................................................ 95
- Mathematical functions.......................................................................... 100
- Multivalue eval functions....................................................................... 106
- Statistical eval functions........................................................................ 115
- Text functions........................................................................................ 117
- Trig and Hyperbolic functions.................................................................. 122

## Statistical and Charting Functions
- Statistical and charting functions............................................................... 130
- Aggregate functions................................................................................. 134
- Event order functions.............................................................................. 154
- Multivalue stats and chart functions....................................................... 159
- Time functions........................................................................................ 162

## Time Format Variables and Modifiers
- Date and time format variables.................................................................. 172
- Time modifiers......................................................................................... 175
Table of Contents

Search Commands

abstract................................................................. 182
accum................................................................. 183
addcoltotals....................................................... 184
addinfo............................................................... 186
addtotals............................................................. 187
analyzefields.................................................... 193
anomalies......................................................... 195
anomalousvalue............................................... 200
anomalydetection............................................... 205
append............................................................... 210
appendcols...................................................... 216
appendpipe....................................................... 218
arules................................................................. 220
associate........................................................... 222
audit................................................................. 227
autoregress...................................................... 228
awssnsalert...................................................... 230
bin................................................................. 230
bucket.............................................................. 234
bucketdir........................................................ 234
cefout............................................................... 236
chart................................................................. 236
cluster.............................................................. 255
cofilter............................................................ 260
collect............................................................... 261
concurrency.................................................... 265
contingency..................................................... 270
convert............................................................. 275
correlate.......................................................... 281
ctable............................................................... 283
datamodel....................................................... 284
datamodelsimple.............................................. 289
dbinspect........................................................ 289
dedup............................................................... 295
delete............................................................... 299
delta................................................................. 302
diff................................................................. 308
erex............................................................... 309
Table of Contents

Search Commands

eval ................................................................. 313
eventcount ...................................................... 327
eventstats ....................................................... 330
extract ............................................................ 334
fieldformat ..................................................... 337
fields ............................................................. 342
fieldsummary .................................................. 345
filldown ........................................................... 347
fillnull ............................................................ 348
findtypes ....................................................... 350
folderize ......................................................... 351
foreach .......................................................... 354
format ............................................................. 357
from .............................................................. 361
gauge ............................................................. 364
gentimes ........................................................ 367
geom .............................................................. 370
geomfilter ..................................................... 376
geostats ........................................................ 378
head .............................................................. 384
highlight ....................................................... 387
history ........................................................... 388
iconify ........................................................... 391
input ............................................................. 393
inputcsv ......................................................... 394
inputintelligence ........................................... 397
inputlookup ................................................... 398
iplocation ...................................................... 401
join .............................................................. 407
kmeans .......................................................... 412
kvform .......................................................... 415
loadjob .......................................................... 418
localize ........................................................ 420
localop .......................................................... 422
lookup .......................................................... 423
makecontinuous ............................................ 429
makemv ........................................................ 431
makeresults ................................................... 434
### Table of Contents

#### Search Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>437</td>
</tr>
<tr>
<td>mcollect</td>
<td>439</td>
</tr>
<tr>
<td>metadata</td>
<td>443</td>
</tr>
<tr>
<td>metasearch</td>
<td>448</td>
</tr>
<tr>
<td>meventcollect</td>
<td>450</td>
</tr>
<tr>
<td>mstats</td>
<td>453</td>
</tr>
<tr>
<td>multikv</td>
<td>462</td>
</tr>
<tr>
<td>multiselect</td>
<td>465</td>
</tr>
<tr>
<td>mvcombine</td>
<td>467</td>
</tr>
<tr>
<td>mvexpand</td>
<td>472</td>
</tr>
<tr>
<td>nomv</td>
<td>475</td>
</tr>
<tr>
<td>outlier</td>
<td>476</td>
</tr>
<tr>
<td>outputcsv</td>
<td>478</td>
</tr>
<tr>
<td>outputlookup</td>
<td>482</td>
</tr>
<tr>
<td>outputtext</td>
<td>486</td>
</tr>
<tr>
<td>overlap</td>
<td>487</td>
</tr>
<tr>
<td>pivot</td>
<td>489</td>
</tr>
<tr>
<td>predict</td>
<td>493</td>
</tr>
<tr>
<td>rangemap</td>
<td>502</td>
</tr>
<tr>
<td>rare</td>
<td>504</td>
</tr>
<tr>
<td>redistribute</td>
<td>506</td>
</tr>
<tr>
<td>regex</td>
<td>515</td>
</tr>
<tr>
<td>relevancy</td>
<td>517</td>
</tr>
<tr>
<td>reltime</td>
<td>518</td>
</tr>
<tr>
<td>rename</td>
<td>519</td>
</tr>
<tr>
<td>replace</td>
<td>522</td>
</tr>
<tr>
<td>rest</td>
<td>524</td>
</tr>
<tr>
<td>return</td>
<td>526</td>
</tr>
<tr>
<td>reverse</td>
<td>529</td>
</tr>
<tr>
<td>rex</td>
<td>530</td>
</tr>
<tr>
<td>rtorder</td>
<td>535</td>
</tr>
<tr>
<td>run</td>
<td>536</td>
</tr>
<tr>
<td>savedsearch</td>
<td>536</td>
</tr>
<tr>
<td>script</td>
<td>538</td>
</tr>
<tr>
<td>scrub</td>
<td>540</td>
</tr>
<tr>
<td>search</td>
<td>542</td>
</tr>
<tr>
<td>searchtxn</td>
<td>552</td>
</tr>
<tr>
<td>selfjoin</td>
<td>554</td>
</tr>
</tbody>
</table>
# Table of Contents

## Search Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendemail</td>
<td>560</td>
</tr>
<tr>
<td>set</td>
<td>566</td>
</tr>
<tr>
<td>setfields</td>
<td>569</td>
</tr>
<tr>
<td>sichart</td>
<td>570</td>
</tr>
<tr>
<td>sirare</td>
<td>571</td>
</tr>
<tr>
<td>sistats</td>
<td>573</td>
</tr>
<tr>
<td>sitimechart</td>
<td>575</td>
</tr>
<tr>
<td>sitop</td>
<td>576</td>
</tr>
<tr>
<td>snowincident</td>
<td>579</td>
</tr>
<tr>
<td>snowincidentstream</td>
<td>579</td>
</tr>
<tr>
<td>snowevent</td>
<td>580</td>
</tr>
<tr>
<td>snoweventstream</td>
<td>580</td>
</tr>
<tr>
<td>sort</td>
<td>580</td>
</tr>
<tr>
<td>spath</td>
<td>585</td>
</tr>
<tr>
<td>stats</td>
<td>591</td>
</tr>
<tr>
<td>strcat</td>
<td>607</td>
</tr>
<tr>
<td>streamstats</td>
<td>609</td>
</tr>
<tr>
<td>table</td>
<td>619</td>
</tr>
<tr>
<td>tags</td>
<td>623</td>
</tr>
<tr>
<td>tail</td>
<td>625</td>
</tr>
<tr>
<td>timechart</td>
<td>626</td>
</tr>
<tr>
<td>timewrap</td>
<td>645</td>
</tr>
<tr>
<td>top</td>
<td>648</td>
</tr>
<tr>
<td>transaction</td>
<td>652</td>
</tr>
<tr>
<td>transpose</td>
<td>664</td>
</tr>
<tr>
<td>trendline</td>
<td>670</td>
</tr>
<tr>
<td>tscollect</td>
<td>672</td>
</tr>
<tr>
<td>tstats</td>
<td>674</td>
</tr>
<tr>
<td>typeahead</td>
<td>684</td>
</tr>
<tr>
<td>typelearner</td>
<td>686</td>
</tr>
<tr>
<td>typer</td>
<td>687</td>
</tr>
<tr>
<td>union</td>
<td>689</td>
</tr>
<tr>
<td>uniq</td>
<td>694</td>
</tr>
<tr>
<td>untable</td>
<td>695</td>
</tr>
<tr>
<td>where</td>
<td>699</td>
</tr>
<tr>
<td>x11</td>
<td>704</td>
</tr>
<tr>
<td>xmlkv</td>
<td>706</td>
</tr>
<tr>
<td>xmlunescape</td>
<td>707</td>
</tr>
</tbody>
</table>
# Table of Contents

## Search Commands
- xpath.......................................................................................................708
- xsDisplayConcept...................................................................................711
- xsDisplayContext....................................................................................711
- xsFindBestConcept.................................................................................711
- xsListConcepts........................................................................................711
- xsListContexts.........................................................................................712
- xsUpdateDDContext...............................................................................712
- xsWhere..................................................................................................712
- xyseries...................................................................................................712
- 3rd party custom commands...................................................................714

## Internal Commands.........................................................................................715
- About internal commands.......................................................................715
- collapse...................................................................................................715
- dump.......................................................................................................716
- findkeywords...........................................................................................718
- mcatalog..................................................................................................719
- noop........................................................................................................725
- runshellscript...........................................................................................730
- sendalert.................................................................................................731

## Search in the CLI..............................................................................................734
- About searches in the CLI.......................................................................734
- Syntax for searches in the CLI.................................................................735
Introduction

Welcome to the Search Reference

This manual is a reference guide for the Search Processing Language (SPL). In this manual you will find a catalog of the search commands with complete syntax, descriptions, and examples. Additionally, this manual includes quick reference information about the categories of commands, the functions you can use with commands, and how SPL relates to SQL.

Getting Started

If you are new to Splunk software and searching, start with the Search Tutorial. This tutorial introduces you to the Search & Reporting application. The tutorial guides you through uploading data to your Splunk deployment, searching your data, and building simple charts, reports, and dashboards.

After you complete the Search Tutorial, and before you start using Splunk software on your own data you should:

- Add data to your Splunk instance. See Getting Data In.
- Understand how indexing works and how data is processed. See Managing Indexers and Clusters of Indexers.
- Learn about fields and knowledge objects, such as hosts, source types, and event types. See the Knowledge Manager Manual.

Search Manual


- Types of searches
- Retrieving events
- Specifying time ranges
- Optimizing searches
- Using subsearches
- Creating statistical tables and charts
- Grouping and correlating events
- Predicting future events
Quick Reference Information

The Quick Reference Guide contains:

- Explanations about Splunk features
- Common search commands
- Tips on optimizing searches
- Functions for the `eval` and `stats` commands
- Search examples
- Regular expressions
- Formats for converting strings into timestamps

Command categories

The search commands by category topic organizes the commands by the type of action that the command performs.

For example, commands in the `reporting` category, are used to build transforming searches. Reporting commands return statistical data tables that are required for charts and other kinds of data visualizations.

This topic contains a brief description of each command along with a link to the details about the command in the Search Commands section of this manual.

Command syntax

Before you continue, see Understanding SPL syntax for the conventions and rules used in this manual.

Understanding SPL syntax

The following sections describe the syntax used for the Splunk SPL commands. For additional information about using keywords, phrases, wildcards, and regular expressions, see Search command primer.

Required and optional arguments

SPL commands consist of required and optional arguments.
• Required arguments are shown in angle brackets < >.
• Optional arguments are enclosed in square brackets [ ].

Consider this command syntax:

    bin [<bins-options>...] <field> [AS <newfield>]

The required argument is <field>. To use this command, at a minimum you must specify bin <field>.

The optional arguments are [<bins-options>...] and [AS <newfield>].

**User input arguments**

Consider this command syntax:

    replace (<wc-string> WITH <wc-string>)... [IN <field-list>]

The user input arguments are: <wc-string> and <field-list>.

**Repeating arguments**

Some arguments can be specified multiple times. The syntax displays ellipsis ... to specify which part of an argument can be repeated. The ellipsis always appear immediately after the part of the syntax that you can repeat.

Consider this command:

    convert [timeformat=string] (<convert-function> [AS <field>])...

The required argument is <convert-function>, with an option to specify a field with the [AS <field>] clause.

Notice the ellipsis at the end of the syntax, just after the close parenthesis. In this example, the syntax that is inside the parenthesis can be repeated <convert-function> [AS <field>].

In the following syntax, you can repeat the <bins-options>....

    bin [<bins-options>...] <field> [AS <newfield>]
Grouped arguments

Sometimes the syntax must display arguments as a group to show that the set of arguments are used together. Parenthesis ( ) are used to group arguments.

For example in this syntax:

\[
\text{replace (\text{<wc-string>} \text{ WITH } \text{<wc-string>})... [IN \text{ <field-list>}]}
\]

The grouped argument is  \((\text{<wc-string>} \text{ WITH } \text{<wc-string>})...\). This is a required set of arguments that you can repeat multiple times.

Keywords

Many commands use keywords with some of the arguments or options. Examples of keywords include:

- AS
- BY
- OVER
- WHERE

You can specify these keywords in uppercase or lowercase in your search. However, for readability, the syntax in the Splunk documentation uses uppercase on all keywords.

Quoted elements

If an element is in quotation marks, you must include that element in your search. The most common quoted elements are parenthesis.

Consider the syntax for the chart command:

\[
\text{chart} [\text{<chart-options>}] [\text{agg=<stats-agg-term>}] \\
( \text{<stats-agg-term>} | \text{<sparkline-agg-term>} | "("\text{<eval-expression}>")") \\
... \text{[ BY \text{<row-split> \text{<column-split>}] | [ OVER \text{<row-split>} ] [BY} \\
\text{<column-split>] ]}
\]

There are quotation marks on the parenthesis surrounding the \text{<eval-expression>}. This means that you must enclose the \text{<eval-expression>} in parenthesis in your search.
In the following search example, the `<eval-expression>` is `avg(size)/max(delay)` and is enclosed in parenthesis.

... | chart eval(avg(size)/max(delay)) AS ratio BY host user

**Argument order**

In the command syntax, the command arguments are presented in the order in which the arguments are meant to be used.

In the descriptions of the arguments, the **Required arguments** and **Optional argument** sections, the arguments are listed alphabetically. For each argument, there is a **Syntax** and **Description**. Additionally, for Optional arguments, there might be a **Default**.

**Data types**

The nomenclature used for the data types in SPL syntax are described in the following table.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;bool&gt;</code></td>
<td>boolean</td>
<td>Use <code>true</code> or <code>false</code>. Other variations are accepted. For example, for <code>true</code> you can also use 't', 'T', 'TRUE', 'yes', or the number one ( 1 ). For <code>false</code> you can also specify 'no', the number zero ( 0 ), and variations of the word <code>false</code>, similar to the variations of the word <code>true</code>.</td>
</tr>
<tr>
<td><code>&lt;field&gt;</code></td>
<td>A field name. You cannot specify a wildcard for the field name.</td>
<td>See <code>&lt;wc-field&gt;</code>.</td>
</tr>
<tr>
<td><code>&lt;int&gt;</code> or <code>&lt;integer&gt;</code></td>
<td>An integer that can be a positive or negative value.</td>
<td>Sometimes referred to as a &quot;signed&quot; integer. See <code>&lt;unsigned int&gt;</code>.</td>
</tr>
<tr>
<td><code>&lt;string&gt;</code></td>
<td>string</td>
<td>See <code>&lt;wc-string&gt;</code>.</td>
</tr>
<tr>
<td><code>&lt;unsigned int&gt;</code></td>
<td>unsigned integer</td>
<td>An unsigned integer must be positive value. Unsigned integers can be larger numbers than signed integers.</td>
</tr>
<tr>
<td><code>&lt;wc-field&gt;</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A field name or a partial name with a wildcard character to specify multiple, similarly named fields.

Use the asterisk ( * ) character as the wildcard character.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wc-string&gt;</td>
<td>A string value or partial string value with a wildcard character.</td>
<td>Use the asterisk ( * ) character as the wildcard character.</td>
</tr>
</tbody>
</table>

**Boolean operators**

When a boolean operator is included in the syntax of a command, you must always specify the operator in uppercase. Boolean operators include:

- AND
- OR
- NOT

To learn more about the order in which boolean expressions are evaluated, along with some examples, see Boolean expressions in the *Search Manual*.

To learn more about the the NOT operator, see Difference between NOT and != in the *Search Manual*.

**BY clauses**

A <by-clause> and a <split-by-clause> are not the same argument.

When you use a <by-clause>, one row is returned for each distinct value <by-clause> field. A <by-clause> displays each unique item in a separate row. Think of the <by-clause> as a grouping.

The <split-by-clause> displays each unique item in a separate column. Think of the <split-by-clause> as a splitting or dividing.

Wildcard characters ( * ) are not accepted in BY clauses.
**Fields and wildcard fields**

When the syntax contains `<field>` you specify a field name from your events.

Consider this syntax:

```
bin [<bins-options>...] <field> [AS <newfield>]
```

The `<field>` argument is required. You can specify that the field displays a different name in the search results by using the `[AS <newfield>]` argument. This argument is optional.

For example, if the field is `categoryId` and you want the field to be named `CategoryID` in the output, you would specify:

```
categoryId AS CategoryID
```

The `<wc-field>` argument indicates that you can use wild card characters when specifying field names. For example, if you have a set of fields that end with "log" you can specify `*log` to return all of those fields.

If you use a wild card character in the middle of a value, especially as a wild card for punctuation, the results might be unpredictable.

**See also**

In the *Search Manual*:

- Anatomy of a search
- Wildcards
- Field expressions
- Quotes and escaping characters

**How to use this manual**

This manual serves as a reference guide for the Splunk user who is looking for a catalog of the search commands with complete syntax, descriptions, and examples for usage.
Quick Reference Information

Functions

Command topics

Each search command topic contains the following sections: Description, Syntax, Examples, and See also. Many of the command topics also have a Usage section.

Description
Describes what the command is used for. This section might include details about how to use the command. For more complex commands, there might be a separate Usage section.

Syntax
The syntax includes the complete syntax for each search command, and a description for each argument. Some commands have arguments that have a set of options that you can specify. Each of these sets of options follow the argument descriptions.

Required arguments
Displays the syntax and describes the required arguments.

Optional arguments
Displays the syntax and describes the optional arguments. Default values, if applicable, are also listed.

Usage
Contains additional information about using the command.

Examples
This section includes examples of how to use the command.

See also
This section contains links to all related or similar commands.

Command syntax conventions
The command arguments are presented in the syntax in the order in which the arguments are meant to be used.
Arguments are either Required or Optional and are listed alphabetically under their respective subheadings. For each argument, there are Syntax and Description sections. Additionally, there might be other sections, such as Default that provide information about the argument.

See Understanding SPL syntax.

Formatting conventions

*Italic*

When referring to another manual in the set of Splunk documentation, the name of the manual appears in italic.
Quick Reference

Splunk Quick Reference Guide

The Splunk Quick Reference Guide is a six-page reference card that provides fundamental search concepts, commands, functions, and examples. This guide is available online as a PDF file.

Note: The examples in this quick reference use a leading ellipsis (...) to indicate that there is a search before the pipe operator. A leading pipe indicates that the search command is a generating command and prevents the command-line interface and Splunk Web from prepending the search command to your search.

See also

- Search commands by category

Splunk Answers

If you cannot find what you are looking for in this search language reference, check out Splunk Answers and see what questions and answers other Splunk users have about the search language.

Command quick reference

The table below lists all of the search commands in alphabetical order. There is a short description of the command and links to related commands. For the complete syntax, usage, and detailed examples, click the command name to display the specific topic for that command.

Some of these commands share functions. For a list of the functions with descriptions and examples, see Evaluation functions and Statistical and charting functions.

If you don't find a command in the table, that command might be part of a third-party app or add-on. For information about commands contributed by apps and add-ons, see the documentation on Splunkbase.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>abstract</td>
<td>Produces a summary of each search result.</td>
</tr>
<tr>
<td>accum</td>
<td>Keeps a running total of the specified numeric field.</td>
</tr>
<tr>
<td>addcoltotals</td>
<td>Computes an event that contains sum of all numeric fields for previous events.</td>
</tr>
<tr>
<td>addinfo</td>
<td>Add fields that contain common information about the current search.</td>
</tr>
<tr>
<td>addtotals</td>
<td>Computes the sum of all numeric fields for each result.</td>
</tr>
<tr>
<td>analyzefields</td>
<td>Analyze numerical fields for their ability to predict another discrete field.</td>
</tr>
<tr>
<td>anomalies</td>
<td>Computes an &quot;unexpectedness&quot; score for an event.</td>
</tr>
<tr>
<td>anomalousvalue</td>
<td>Finds and summarizes irregular, or uncommon, search results.</td>
</tr>
<tr>
<td>anomalydetection</td>
<td>Identifies anomalous events by computing a probability for each event and then detecting unusually small probabilities.</td>
</tr>
<tr>
<td>append</td>
<td>Appends subsearch results to current results.</td>
</tr>
<tr>
<td>appendcols</td>
<td>Appends the fields of the subsearch results to current results, first results to first result, second to second, etc.</td>
</tr>
<tr>
<td>appendpipe</td>
<td>Appends the result of the subpipeline applied to the current result set to results.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>arules</td>
<td>Finds association rules between field values.</td>
</tr>
<tr>
<td>associate</td>
<td>Identifies correlations between fields.</td>
</tr>
<tr>
<td>correlate</td>
<td>Calculates the correlation between different fields.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>datamodel</td>
<td>Examine data model dataset and search a data model dataset.</td>
</tr>
<tr>
<td>dbinspect</td>
<td>Returns information about the specified index.</td>
</tr>
<tr>
<td>dedup</td>
<td>Removes subsequent results that match a specified criteria.</td>
</tr>
<tr>
<td>delete</td>
<td>Delete specific events or search results.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>delta</td>
<td>Computes the difference in field value between nearby results.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>diff</td>
<td>Returns the difference between two search results.</td>
</tr>
<tr>
<td>erex</td>
<td>Allows you to specify example or counter example values to automatically extract fields that have similar values.</td>
</tr>
<tr>
<td>eval</td>
<td>Calculates an expression and puts the value into a field. See also, Evaluation functions.</td>
</tr>
<tr>
<td>eventcount</td>
<td>Returns the number of events in an index.</td>
</tr>
<tr>
<td>eventstats</td>
<td>Adds summary statistics to all search results.</td>
</tr>
<tr>
<td>extract (kv)</td>
<td>Extracts field-value pairs from search results.</td>
</tr>
<tr>
<td>fieldformat</td>
<td>Expresses how to render a field at output time without changing the underlying value.</td>
</tr>
<tr>
<td>fields</td>
<td>Removes fields from search results.</td>
</tr>
<tr>
<td>fieldsummary</td>
<td>Generates summary information for all or a subset of the fields.</td>
</tr>
<tr>
<td>filldown</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fillnull</td>
<td>Replaces null values with a specified value.</td>
</tr>
<tr>
<td>findtypes</td>
<td>Generates a list of suggested event types.</td>
</tr>
<tr>
<td>typer</td>
<td>Creates a higher-level grouping, such as replacing filenames with directories.</td>
</tr>
<tr>
<td>foreach</td>
<td>Run a templatized streaming subsearch for each field in a wildcarded field list.</td>
</tr>
<tr>
<td>eval</td>
<td>Takes the results of a subsearch and formats them into a single result.</td>
</tr>
<tr>
<td>from</td>
<td>Retrieves data from a dataset, such as a data model dataset, a CSV lookup, a KV Store lookup, a saved search, or a table dataset.</td>
</tr>
<tr>
<td>gauge</td>
<td>Transform results into a format suitable for display by the Gauge chart types.</td>
</tr>
<tr>
<td>gentimes</td>
<td>Generates time-range results.</td>
</tr>
<tr>
<td>geom</td>
<td>Adds a field, named &quot;geom&quot;, to each event. This field contains geographic data structures for polygon geometry in JSON and is used for the choropleth map visualization.</td>
</tr>
<tr>
<td>geomfilter</td>
<td>Accepts two points that specify a bounding box for clipping a choropleth map. Points that fall outside of the bounding box are filtered out.</td>
</tr>
<tr>
<td>geostats</td>
<td>Generate statistics which are clustered into geographical bins to be rendered on a world map.</td>
</tr>
<tr>
<td>head</td>
<td>Returns the first number n of specified results.</td>
</tr>
<tr>
<td>highlight</td>
<td>Highlights the specified terms.</td>
</tr>
<tr>
<td>history</td>
<td>Returns a history of searches formatted as an events list or as a table.</td>
</tr>
<tr>
<td>iconify</td>
<td>Displays a unique icon for each different value in the list of fields that you specify.</td>
</tr>
<tr>
<td>input</td>
<td>Add or disable sources.</td>
</tr>
<tr>
<td>inputcsv</td>
<td>Loads search results from the specified CSV file.</td>
</tr>
<tr>
<td>inputlookup</td>
<td>Loads search results from a specified static lookup table.</td>
</tr>
<tr>
<td>iplocation</td>
<td>Extracts location information from IP addresses.</td>
</tr>
<tr>
<td>join</td>
<td>Combine the results of a subsearch with the results of a main search.</td>
</tr>
<tr>
<td>kmeans</td>
<td>Performs k-means clustering on selected fields.</td>
</tr>
<tr>
<td>kvform</td>
<td>Extracts values from search results, using a form template.</td>
</tr>
<tr>
<td>loadjob</td>
<td>Loads events or results of a previously completed search job.</td>
</tr>
<tr>
<td>localize</td>
<td>Returns a list of the time ranges in which the search results were found.</td>
</tr>
<tr>
<td>localop</td>
<td>Run subsequent commands, that is all commands following this, locally and not on remote peers.</td>
</tr>
<tr>
<td>lookup</td>
<td>Explicitly invokes field value lookups.</td>
</tr>
<tr>
<td>makecontinuous</td>
<td>Makes a field that is supposed to be the x-axis continuous (invoked by chart/timechart)</td>
</tr>
<tr>
<td>makemv</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>outlier</td>
<td>Removes outlying numerical values.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>outputcsv</td>
<td>Outputs search results to a specified CSV file.</td>
</tr>
<tr>
<td>outputlookup</td>
<td>Writes search results to the specified static lookup table.</td>
</tr>
<tr>
<td>outputtext</td>
<td>Outputs the raw text field (_raw) of results into the _xml field.</td>
</tr>
<tr>
<td>overlap</td>
<td>Finds events in a summary index that overlap in time or have missed events.</td>
</tr>
<tr>
<td>pivot</td>
<td>Run pivot searches against a particular data model dataset.</td>
</tr>
<tr>
<td>predict</td>
<td>Enables you to use time series algorithms to predict future values of fields.</td>
</tr>
<tr>
<td>rangemap</td>
<td>Sets RANGE field to the name of the ranges that match.</td>
</tr>
<tr>
<td>rare</td>
<td>Displays the least common values of a field.</td>
</tr>
<tr>
<td>regex</td>
<td>Removes results that do not match the specified regular expression.</td>
</tr>
<tr>
<td>relevancy</td>
<td>Calculates how well the event matches the query.</td>
</tr>
<tr>
<td>reltime</td>
<td>Converts the difference between 'now' and '_time' to a human-readable value and adds this value to the field, 'reltime', in your search results.</td>
</tr>
<tr>
<td>rename</td>
<td>Renames a specified field; wildcards can be used to specify multiple fields.</td>
</tr>
<tr>
<td>replace</td>
<td>Replaces values of specified fields with a specified new value.</td>
</tr>
<tr>
<td>rest</td>
<td>Access a REST endpoint and display the returned REST entities as search</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>sort</td>
<td>Sorts search results by the specified fields.</td>
</tr>
<tr>
<td>spath</td>
<td>Provides a straightforward means for extracting fields from structured data formats, XML and JSON.</td>
</tr>
<tr>
<td>stats</td>
<td>Provides statistics, grouped optionally by fields. See also, Statistical and charting functions.</td>
</tr>
<tr>
<td>strcat</td>
<td>Concatenates string values.</td>
</tr>
<tr>
<td>streamstats</td>
<td>Adds summary statistics to all search results in a streaming manner.</td>
</tr>
<tr>
<td>table</td>
<td>Creates a table using the specified fields.</td>
</tr>
<tr>
<td>tags</td>
<td>Annotates specified fields in your search results with tags.</td>
</tr>
<tr>
<td>tail</td>
<td>Returns the last number n of specified results.</td>
</tr>
<tr>
<td>timechart</td>
<td>Create a time series chart and corresponding table of statistics. See also, Statistical and charting functions.</td>
</tr>
<tr>
<td>timewrap</td>
<td>Displays, or wraps, the output of the timechart command so that every timewrap-span range of time is a different series.</td>
</tr>
<tr>
<td>top</td>
<td>Displays the most common values of a field.</td>
</tr>
<tr>
<td>transaction</td>
<td>Groups search results into transactions.</td>
</tr>
<tr>
<td>transpose</td>
<td>Reformats rows of search results as columns.</td>
</tr>
<tr>
<td>trendline</td>
<td>Computes moving averages of fields.</td>
</tr>
<tr>
<td>tscollect</td>
<td>Writes results into tsidx file(s) for later use by tstats command.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tstats</td>
<td>Calculates statistics over tsidx files created with the tscollect command.</td>
</tr>
<tr>
<td>typeahead</td>
<td>Returns typeahead information on a specified prefix.</td>
</tr>
<tr>
<td>typelearner</td>
<td>Generates suggested eventtypes.</td>
</tr>
<tr>
<td>typer</td>
<td>Calculates the eventtypes for the search results.</td>
</tr>
<tr>
<td>union</td>
<td>Merges the results from two or more datasets into one dataset.</td>
</tr>
<tr>
<td>uniq</td>
<td>Removes any search that is an exact duplicate with a previous result.</td>
</tr>
<tr>
<td>untable</td>
<td>Converts results from a tabular format to a format similar to stats output.</td>
</tr>
<tr>
<td></td>
<td>Inverse of xyseries and maketable.</td>
</tr>
<tr>
<td>where</td>
<td>Performs arbitrary filtering on your data. See also, Evaluations functions.</td>
</tr>
<tr>
<td>x11</td>
<td>Enables you to determine the trend in your data by removing the seasonal</td>
</tr>
<tr>
<td></td>
<td>pattern.</td>
</tr>
<tr>
<td>xmlkv</td>
<td>Extracts XML key-value pairs.</td>
</tr>
<tr>
<td></td>
<td>multikv, rex</td>
</tr>
<tr>
<td>xmlunescape</td>
<td>Unescapes XML.</td>
</tr>
<tr>
<td>xpath</td>
<td>Redefines the XML path.</td>
</tr>
<tr>
<td>xyseries</td>
<td>Converts results into a format suitable for graphing.</td>
</tr>
</tbody>
</table>

**Commands by category**

The following tables list all the search commands, categorized by their usage. Some commands fit into more than one category based on the options that you specify.
Correlation

These commands can be used to build correlation searches.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>Appends subsearch results to current results.</td>
</tr>
<tr>
<td>appendcols</td>
<td>Appends the fields of the subsearch results to current results, first result to first result, second to second, etc.</td>
</tr>
<tr>
<td>appendpipe</td>
<td>Appends the result of the subpipeline applied to the current result set to results.</td>
</tr>
<tr>
<td>arules</td>
<td>Finds association rules between field values.</td>
</tr>
<tr>
<td>associate</td>
<td>Identifies correlations between fields.</td>
</tr>
<tr>
<td>contingency, countable, ctable</td>
<td>Builds a contingency table for two fields.</td>
</tr>
<tr>
<td>correlate</td>
<td>Calculates the correlation between different fields.</td>
</tr>
<tr>
<td>diff</td>
<td>Returns the difference between two search results.</td>
</tr>
<tr>
<td>join</td>
<td>Combines the results from the main results pipeline with the results from a subsearch.</td>
</tr>
<tr>
<td>lookup</td>
<td>Explicitly invokes field value lookups.</td>
</tr>
<tr>
<td>selfjoin</td>
<td>Joins results with itself.</td>
</tr>
<tr>
<td>set</td>
<td>Performs set operations (union, diff, intersect) on subsearches.</td>
</tr>
<tr>
<td>stats</td>
<td>Provides statistics, grouped optionally by fields. See Statistical and charting functions.</td>
</tr>
<tr>
<td>transaction</td>
<td>Groups search results into transactions.</td>
</tr>
</tbody>
</table>

Data and indexes

These commands can be used to learn more about your data, add and delete data sources, or manage the data in your summary indexes.

View data

These commands return information about the data you have in your indexes. They do not modify your data or indexes in any way.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit</td>
<td>Returns audit trail information that is stored in the local audit index.</td>
</tr>
<tr>
<td>datamodel</td>
<td>Return information about a data model or data model object.</td>
</tr>
<tr>
<td>dbinspect</td>
<td>Returns information about the specified index.</td>
</tr>
<tr>
<td>eventcount</td>
<td>Returns the number of events in an index.</td>
</tr>
<tr>
<td>metadata</td>
<td>Returns a list of source, sourcetypes, or hosts from a specified index or distributed search peer.</td>
</tr>
<tr>
<td>typeahead</td>
<td>Returns typeahead information on a specified prefix.</td>
</tr>
</tbody>
</table>

**Manage data**

These are some commands you can use to add data sources to or delete specific data from your indexes.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete</td>
<td>Delete specific events or search results.</td>
</tr>
<tr>
<td>input</td>
<td>Add or disable sources.</td>
</tr>
</tbody>
</table>

**Manage summary indexes**

These commands are used to create and manage your summary indexes.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collect, stash</td>
<td>Puts search results into a summary index.</td>
</tr>
<tr>
<td>overlap</td>
<td>Finds events in a summary index that overlap in time or have missed events.</td>
</tr>
<tr>
<td>sichart</td>
<td>Summary indexing version of chart. Computes the necessary information for you to later run a chart search on the summary index.</td>
</tr>
<tr>
<td>sirare</td>
<td>Summary indexing version of rare. Computes the necessary information for you to later run a rare search on the summary index.</td>
</tr>
<tr>
<td>sistats</td>
<td>Summary indexing version of stats. Computes the necessary information for you to later run a stats search on the summary index.</td>
</tr>
<tr>
<td>sitimechart</td>
<td>Summary indexing version of timechart. Computes the necessary information for you to later run a timechart</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>search on the summary index.</td>
<td></td>
</tr>
<tr>
<td>sitop</td>
<td>Summary indexing version of top. Computes the necessary information for you to later run a top search on the summary index.</td>
</tr>
</tbody>
</table>

**Fields**

These are commands you can use to add, extract, and modify fields or field values. The most useful command for manipulating fields is `eval` and its statistical and charting functions.

**Add fields**

Use these commands to add new fields.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accum</td>
<td>Keeps a running total of the specified numeric field.</td>
</tr>
<tr>
<td>addinfo</td>
<td>Add fields that contain common information about the current search.</td>
</tr>
<tr>
<td>addtotals</td>
<td>Computes the sum of all numeric fields for each result.</td>
</tr>
<tr>
<td>delta</td>
<td>Computes the difference in field value between nearby results.</td>
</tr>
<tr>
<td>eval</td>
<td>Calculates an expression and puts the value into a field. See also, evaluation functions.</td>
</tr>
<tr>
<td>iplocation</td>
<td>Adds location information, such as city, country, latitude, longitude, and so on, based on IP addresses.</td>
</tr>
<tr>
<td>lookup</td>
<td>For configured lookup tables, explicitly invokes the field value lookup and adds fields from the lookup table to the events.</td>
</tr>
<tr>
<td>multikv</td>
<td>Extracts field-values from table-formatted events.</td>
</tr>
<tr>
<td>rangemap</td>
<td>Sets RANGE field to the name of the ranges that match.</td>
</tr>
<tr>
<td>relevancy</td>
<td>Adds a relevancy field, which indicates how well the event matches the query.</td>
</tr>
<tr>
<td>strcat</td>
<td>Concatenates string values and saves the result to a specified field.</td>
</tr>
</tbody>
</table>
**Extract fields**

These commands provide different ways to extract new fields from search results.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>erex</td>
<td>Allows you to specify example or counter example values to automatically extract fields that have similar values.</td>
</tr>
<tr>
<td>extract, kv</td>
<td>Extracts field-value pairs from search results.</td>
</tr>
<tr>
<td>kvform</td>
<td>Extracts values from search results, using a form template.</td>
</tr>
<tr>
<td>rex</td>
<td>Specify a Perl regular expression named groups to extract fields while you search.</td>
</tr>
<tr>
<td>spath</td>
<td>Provides a straightforward means for extracting fields from structured data formats, XML and JSON.</td>
</tr>
<tr>
<td>xmlkv</td>
<td>Extracts XML key-value pairs.</td>
</tr>
</tbody>
</table>

**Modify fields and field values**

Use these commands to modify fields or their values.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>convert</td>
<td>Converts field values into numerical values.</td>
</tr>
<tr>
<td>filldown</td>
<td>Replaces NULL values with the last non-NULL value.</td>
</tr>
<tr>
<td>fillnull</td>
<td>Replaces null values with a specified value.</td>
</tr>
<tr>
<td>makemv</td>
<td>Change a specified field into a multivalue field during a search.</td>
</tr>
<tr>
<td>nomv</td>
<td>Changes a specified multivalue field into a single-value field at search time.</td>
</tr>
<tr>
<td>reltime</td>
<td>Converts the difference between 'now' and '_time' to a human-readable value and adds this value to the field, 'reltime', in your search results.</td>
</tr>
<tr>
<td>rename</td>
<td>Renames a specified field. Use wildcards to specify multiple fields.</td>
</tr>
<tr>
<td>replace</td>
<td>Replaces values of specified fields with a specified new value.</td>
</tr>
</tbody>
</table>
Find anomalies

These commands are used to find anomalies in your data. Either search for uncommon or outlying events and fields or cluster similar events together.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>analyzefields, af</code></td>
<td>Analyze numerical fields for their ability to predict another discrete field.</td>
</tr>
<tr>
<td><code>anomalies</code></td>
<td>Computes an &quot;unexpectedness&quot; score for an event.</td>
</tr>
<tr>
<td><code>anomalousvalue</code></td>
<td>Finds and summarizes irregular, or uncommon, search results.</td>
</tr>
<tr>
<td><code>anomalydetection</code></td>
<td>Identifies anomalous events by computing a probability for each event and then detecting unusually small probabilities.</td>
</tr>
<tr>
<td><code>cluster</code></td>
<td>Clusters similar events together.</td>
</tr>
<tr>
<td><code>kmeans</code></td>
<td>Performs k-means clustering on selected fields.</td>
</tr>
<tr>
<td><code>outlier</code></td>
<td>Removes outlying numerical values.</td>
</tr>
<tr>
<td><code>rare</code></td>
<td>Displays the least common values of a field.</td>
</tr>
</tbody>
</table>

Geographic and location

These commands add geographical information to your search results.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iplocation</code></td>
<td>Returns location information, such as city, country, latitude, longitude, and so on, based on IP addresses.</td>
</tr>
<tr>
<td><code>geom</code></td>
<td>Adds a field, named &quot;geom&quot;, to each event. This field contains geographic data structures for polygon geometry in JSON and is used for choropleth map visualization. This command requires an external lookup with <code>external_type=geo</code> to be installed.</td>
</tr>
<tr>
<td><code>geomfilter</code></td>
<td>Accepts two points that specify a bounding box for clipping choropleth maps. Points that fall outside of the bounding box are filtered out.</td>
</tr>
<tr>
<td><code>geostats</code></td>
<td>Generate statistics which are clustered into geographical bins to be rendered on a world map.</td>
</tr>
</tbody>
</table>
Metrics

These commands work with metrics data.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcollect</td>
<td>Converts events into metric data points and inserts the data points into a metric index on the search head.</td>
</tr>
<tr>
<td>meventcollect</td>
<td>Converts events into metric data points and inserts the data points into a metric index on indexer tier.</td>
</tr>
<tr>
<td>mstats</td>
<td>Calculates visualization-ready statistics for the measurement, metric_name, and dimension fields in metric indexes.</td>
</tr>
</tbody>
</table>

Prediction and trending

These commands predict future values and calculate trendlines that can be used to create visualizations.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict</td>
<td>Enables you to use time series algorithms to predict future values of fields.</td>
</tr>
<tr>
<td>trendline</td>
<td>Computes moving averages of fields.</td>
</tr>
<tr>
<td>x11</td>
<td>Enables you to determine the trend in your data by removing the seasonal pattern.</td>
</tr>
</tbody>
</table>

Reports

These commands are used to build transforming searches. These commands return statistical data tables that are required for charts and other kinds of data visualizations.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addtotals</td>
<td>Computes the sum of all numeric fields for each result.</td>
</tr>
<tr>
<td>autoregress</td>
<td>Prepares your events for calculating the autoregression, or moving average, based on a field that you specify.</td>
</tr>
<tr>
<td>bin, discretize</td>
<td>Puts continuous numerical values into discrete sets.</td>
</tr>
<tr>
<td>chart</td>
<td>Returns results in a tabular output for charting. See also, Statistical and charting functions.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>contingency,</td>
<td>Builds a contingency table for two fields.</td>
</tr>
<tr>
<td>counttable,</td>
<td></td>
</tr>
<tr>
<td>ctable</td>
<td></td>
</tr>
<tr>
<td>correlate</td>
<td>Calculates the correlation between different fields.</td>
</tr>
<tr>
<td>eventcount</td>
<td>Returns the number of events in an index.</td>
</tr>
<tr>
<td>eventstats</td>
<td>Adds summary statistics to all search results.</td>
</tr>
<tr>
<td>gauge</td>
<td>Transforms results into a format suitable for display by the Gauge chart types.</td>
</tr>
<tr>
<td>makecontinuous</td>
<td>Makes a field that is supposed to be the x-axis continuous (invoked by chart/timechart)</td>
</tr>
<tr>
<td>mstats</td>
<td>Calculates statistics for the measurement, metric_name, and dimension fields in metric indexes.</td>
</tr>
<tr>
<td>outlier</td>
<td>Removes outlying numerical values.</td>
</tr>
<tr>
<td>rare</td>
<td>Displays the least common values of a field.</td>
</tr>
<tr>
<td>stats</td>
<td>Provides statistics, grouped optionally by fields. See also, Statistical and charting functions.</td>
</tr>
<tr>
<td>streamstats</td>
<td>Adds summary statistics to all search results in a streaming manner.</td>
</tr>
<tr>
<td>timechart</td>
<td>Create a time series chart and corresponding table of statistics. See also, Statistical and charting functions.</td>
</tr>
<tr>
<td>top</td>
<td>Displays the most common values of a field.</td>
</tr>
<tr>
<td>trendline</td>
<td>Computes moving averages of fields.</td>
</tr>
<tr>
<td>tstats</td>
<td>Performs statistical queries on indexed fields in tsidx files.</td>
</tr>
<tr>
<td>untable</td>
<td>Converts results from a tabular format to a format similar to stats output. Inverse of xyseries and maketable.</td>
</tr>
<tr>
<td>xyseries</td>
<td>Converts results into a format suitable for graphing.</td>
</tr>
</tbody>
</table>

**Results**

These commands can be used to manage search results. For example, you can append one set of results with another, filter more events from the results, reformat the results, and so on.
**Alerting**

Use this command to email the results of a search.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendemail</td>
<td>Emails search results, either inline or as an attachment, to one or more specified email addresses.</td>
</tr>
</tbody>
</table>

**Appending**

Use these commands to append one set of results with another set or to itself.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>Appends subsearch results to current results.</td>
</tr>
<tr>
<td>appendcols</td>
<td>Appends the fields of the subsearch results to current results, first results to first result, second to second, and so on.</td>
</tr>
<tr>
<td>join</td>
<td>SQL-like joining of results from the main results pipeline with the results from the subpipeline.</td>
</tr>
<tr>
<td>selfjoin</td>
<td>Joins results with itself.</td>
</tr>
</tbody>
</table>

**Filtering**

Use these commands to remove more events or fields from your current results.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dedup</td>
<td>Removes subsequent results that match a specified criteria.</td>
</tr>
<tr>
<td>fields</td>
<td>Removes fields from search results.</td>
</tr>
<tr>
<td>from</td>
<td>Retrieves data from a dataset, such as a data model dataset, a CSV lookup, a KV Store lookup, a saved search, or a table dataset.</td>
</tr>
<tr>
<td>mvcombine</td>
<td>Combines events in search results that have a single differing field value into one result with a multivalue field of the differing field.</td>
</tr>
<tr>
<td>regex</td>
<td>Removes results that do not match the specified regular expression.</td>
</tr>
<tr>
<td>searchtxn</td>
<td>Finds transaction events within specified search constraints.</td>
</tr>
<tr>
<td>table</td>
<td>Creates a table using the specified fields.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>uniq</td>
<td>Removes any search that is an exact duplicate with a previous result.</td>
</tr>
<tr>
<td>where</td>
<td>Performs arbitrary filtering on your data. See also, Evaluation functions.</td>
</tr>
</tbody>
</table>

### Formatting

Use these commands to reformat your current results.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>untable</td>
<td>Converts results from a tabular format to a format similar to stats output. Inverse of xyseries and maketable.</td>
</tr>
<tr>
<td>xyseries</td>
<td>Converts results into a format suitable for graphing.</td>
</tr>
</tbody>
</table>

### Generating

Use these commands to generate or return events.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gentimes</td>
<td>Returns results that match a time-range.</td>
</tr>
<tr>
<td>loadjob</td>
<td>Loads events or results of a previously completed search job.</td>
</tr>
<tr>
<td>makeresults</td>
<td>Creates a specified number of empty search results.</td>
</tr>
<tr>
<td>mvexpand</td>
<td>Expands the values of a multivalue field into separate events for each value of the multivalue field.</td>
</tr>
<tr>
<td>savedsearch</td>
<td>Returns the search results of a saved search.</td>
</tr>
<tr>
<td>search</td>
<td>Searches indexes for matching events. This command is implicit at the start of every search pipeline that does not begin with another generating command.</td>
</tr>
</tbody>
</table>

### Grouping

Use these commands to group or classify the current results.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster</td>
<td>Clusters similar events together.</td>
</tr>
<tr>
<td>kmeans</td>
<td>Performs k-means clustering on selected fields.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Expands the values of a multivalue field into separate events for each value of the multivalue field.</td>
</tr>
<tr>
<td>transaction</td>
<td>Groups search results into transactions.</td>
</tr>
<tr>
<td>typelearner</td>
<td>Generates suggested eventtypes.</td>
</tr>
<tr>
<td>typer</td>
<td>Calculates the eventtypes for the search results.</td>
</tr>
</tbody>
</table>

**Reordering**

Use these commands to change the order of the current search results.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>head</td>
<td>Returns the first number n of specified results.</td>
</tr>
<tr>
<td>reverse</td>
<td>Reverses the order of the results.</td>
</tr>
<tr>
<td>sort</td>
<td>Sorts search results by the specified fields.</td>
</tr>
<tr>
<td>tail</td>
<td>Returns the last number N of specified results.</td>
</tr>
</tbody>
</table>

**Reading**

Use these commands to read in results from external files or previous searches.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputcsv</td>
<td>Loads search results from the specified CSV file.</td>
</tr>
<tr>
<td>inputlookup</td>
<td>Loads search results from a specified static lookup table.</td>
</tr>
<tr>
<td>loadjob</td>
<td>Loads events or results of a previously completed search job.</td>
</tr>
</tbody>
</table>

**Writing**

Use these commands to define how to output current search results.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collect, stash</td>
<td>Puts search results into a summary index.</td>
</tr>
<tr>
<td>meventcollect</td>
<td>Converts events into metric data points and inserts the data points into a metric index on indexer tier.</td>
</tr>
<tr>
<td>mcollect</td>
<td>Converts events into metric data points and inserts the data points into a metric index on the search head.</td>
</tr>
<tr>
<td>outputcsv</td>
<td>Outputs search results to a specified CSV file.</td>
</tr>
</tbody>
</table>
### Command Description

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputlookup</td>
<td>Writes search results to the specified static lookup table.</td>
</tr>
<tr>
<td>outputtext</td>
<td>Outputs the raw text field (_raw) of results into the _xml field.</td>
</tr>
<tr>
<td>sendemail</td>
<td>Emails search results, either inline or as an attachment, to one or more specified email addresses.</td>
</tr>
</tbody>
</table>

### Search

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>localop</td>
<td>Run subsequent commands, that is all commands following this, locally and not on a remote peer.</td>
</tr>
<tr>
<td>map</td>
<td>A looping operator, performs a search over each search result.</td>
</tr>
<tr>
<td>redistribute</td>
<td>Invokes parallel reduce search processing to shorten the search runtime of a set of supported SPL commands.</td>
</tr>
<tr>
<td>search</td>
<td>Searches indexes for matching events. This command is implicit at the start of every search pipeline that does not begin with another generating command.</td>
</tr>
<tr>
<td>sendemail</td>
<td>Emails search results, either inline or as an attachment, to one or more specified email addresses.</td>
</tr>
</tbody>
</table>

### Subsearch

These are commands that you can use with **subsearches**.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>Appends subsearch results to current results.</td>
</tr>
<tr>
<td>appendcols</td>
<td>Appends the fields of the subsearch results to current results, first results to first result, second to second, and so on.</td>
</tr>
<tr>
<td>appendpipe</td>
<td>Appends the result of the subpipeline applied to the current result set to results.</td>
</tr>
<tr>
<td>foreach</td>
<td>Runs a templated streaming subsearch for each field in a wildcarded field list.</td>
</tr>
<tr>
<td>format</td>
<td>Takes the results of a subsearch and formats them into a single result.</td>
</tr>
<tr>
<td>join</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>return</code></td>
<td>Specify the values to return from a subsearch.</td>
</tr>
<tr>
<td><code>set</code></td>
<td>Performs set operations (union, diff, intersect) on subsearches.</td>
</tr>
</tbody>
</table>

**Time**

Use these commands to search based on time ranges or add time information to your events.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gentimes</code></td>
<td>Returns results that match a time-range.</td>
</tr>
<tr>
<td><code>localize</code></td>
<td>Returns a list of the time ranges in which the search results were found.</td>
</tr>
<tr>
<td><code>reltime</code></td>
<td>Converts the difference between 'now' and '_time' to a human-readable value and adds this value to the field, 'reltime', in your search results.</td>
</tr>
</tbody>
</table>

**Command types**

There are six broad types for all of the search commands: distributable streaming, centralized streaming, transforming, generating, orchestrating and dataset processing. These types are not mutually exclusive. A command might be streaming or transforming, and also generating.

The following tables list the commands that fit into each of these types. For detailed explanations about each of the types, see Types of commands in the Search Manual.

**Streaming commands**

A **streaming command** operates on each event as the event is returned by a search.

- A distributable streaming command runs on the indexer or the search head, depending on where in the search the command is invoked. Distributable streaming commands can be applied to subsets of indexed data in a parallel manner.
A centralized streaming command applies a transformation to each event returned by a search. Unlike distributable streaming commands, a centralized streaming command only works on the search head.

<table>
<thead>
<tr>
<th>Command</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>addinfo</td>
<td>Distributable streaming</td>
</tr>
<tr>
<td>addtotals</td>
<td>Distributable streaming. A transforming command when used to calculate column totals (not row totals).</td>
</tr>
<tr>
<td>anomalydetection</td>
<td></td>
</tr>
<tr>
<td>append</td>
<td></td>
</tr>
<tr>
<td>arules</td>
<td></td>
</tr>
<tr>
<td>autoregress</td>
<td>Centralized streaming.</td>
</tr>
<tr>
<td>bin</td>
<td>Streaming if specified with the span argument.</td>
</tr>
<tr>
<td>bucketdir</td>
<td></td>
</tr>
<tr>
<td>cluster</td>
<td>Streaming in some modes.</td>
</tr>
<tr>
<td>convert</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>dedup</td>
<td>Streaming in some modes.</td>
</tr>
<tr>
<td>eval</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>extract</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>fieldformat</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>fields</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>fillnull</td>
<td>Distributable streaming when a field-list is specified. A dataset processing command when no field-list is specified.</td>
</tr>
<tr>
<td>head</td>
<td>Centralized streaming.</td>
</tr>
<tr>
<td>highlight</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>iconify</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>iplocation</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>join</td>
<td>Centralized streaming, if there is a defined set of fields to join to. A dataset processing command when no field-list is specified.</td>
</tr>
<tr>
<td>lookup</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Notes</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>makemv</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>multikv</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>mvexpand</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>nomv</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>rangemap</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>regex</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>reltime</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>rename</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>replace</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>rex</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>search</td>
<td>Distributable streaming if used further down the search pipeline. A generating command when it is the first command in the search.</td>
</tr>
<tr>
<td>spath</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>strcat</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>streamstats</td>
<td>Centralized streaming.</td>
</tr>
<tr>
<td>tags</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>transaction</td>
<td>Centralized streaming.</td>
</tr>
<tr>
<td>typer</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>where</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>unsetable</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>xmlkv</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>xmlunescape</td>
<td></td>
</tr>
<tr>
<td>xpath</td>
<td>Distributable streaming.</td>
</tr>
<tr>
<td>xyseries</td>
<td>Distributable streaming if the argument grouped=false is specified, which is the default. Otherwise a transforming command.</td>
</tr>
</tbody>
</table>
### Generating commands

A **generating command** generates events or reports from one or more indexes without transforming the events.

<table>
<thead>
<tr>
<th>Command</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>datamodel</td>
<td>Report-generating</td>
</tr>
<tr>
<td>dbinspect</td>
<td>Report-generating.</td>
</tr>
<tr>
<td>eventcount</td>
<td>Report-generating.</td>
</tr>
<tr>
<td>from</td>
<td>Can be either report-generating or event-generating depending on the search or knowledge object that is referenced by the command.</td>
</tr>
<tr>
<td>gentimes</td>
<td>Event-generating.</td>
</tr>
<tr>
<td>inputcsv</td>
<td>Event-generating (centralized).</td>
</tr>
<tr>
<td>Inputlookup</td>
<td>Event-generating (centralized) when <code>append=false</code>, which is the default.</td>
</tr>
<tr>
<td>loadjob</td>
<td>Event-generating (centralized).</td>
</tr>
<tr>
<td>makeresults</td>
<td>Report-generating.</td>
</tr>
<tr>
<td>metadata</td>
<td>Report-generating. Although metadata fetches data from all peers, any command run after it runs only on the search head.</td>
</tr>
<tr>
<td>mstats</td>
<td>Report-generating, except when <code>append=true</code> is specified.</td>
</tr>
<tr>
<td>multisearch</td>
<td>Event-generating.</td>
</tr>
<tr>
<td>pivot</td>
<td>Report-generating.</td>
</tr>
<tr>
<td>rest</td>
<td></td>
</tr>
<tr>
<td>search</td>
<td>Event-generating (distributable) when the first command in the search, which is the default. A streaming (distributable) command if used later in the search pipeline.</td>
</tr>
<tr>
<td>searchtxn</td>
<td>Event-generating.</td>
</tr>
<tr>
<td>set</td>
<td>Event-generating.</td>
</tr>
<tr>
<td>tstats</td>
<td>Report-generating (distributable) when <code>prestats=true</code>. When <code>prestats=false, tstats</code> is event-generating.</td>
</tr>
</tbody>
</table>
**Transforming commands**

A **transforming command** orders the results into a data table. The command "transforms" the specified cell values for each event into numerical values for statistical purposes.

In earlier versions of Splunk software, transforming commands were referred to as reporting commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>addtotals</td>
<td>Transforming when used to calculate column totals (not row totals). A distributable <strong>streaming</strong> command when used to calculate row totals, which is the default.</td>
</tr>
<tr>
<td>chart</td>
<td></td>
</tr>
<tr>
<td>cofilter</td>
<td></td>
</tr>
<tr>
<td>contingency</td>
<td></td>
</tr>
<tr>
<td>history</td>
<td></td>
</tr>
<tr>
<td>makecontinuous</td>
<td></td>
</tr>
<tr>
<td>mvcombine</td>
<td></td>
</tr>
<tr>
<td>rare</td>
<td></td>
</tr>
<tr>
<td>stats</td>
<td></td>
</tr>
<tr>
<td>table</td>
<td></td>
</tr>
<tr>
<td>timechart</td>
<td></td>
</tr>
<tr>
<td>top</td>
<td></td>
</tr>
<tr>
<td>xyseries</td>
<td>Transforming if <code>grouped=true</code>. A <strong>streaming</strong> (distributable) command when used to <code>grouped=false</code>, which is the default setting.</td>
</tr>
</tbody>
</table>

**Orchestrating commands**

**Orchestrating commands** control some aspect of how a search is processed. They do not directly affect the final result set of the search. For example, you might apply an orchestrating command to a search to enable or disable a search optimization that helps the overall search complete faster.

<table>
<thead>
<tr>
<th>Command</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>localop</td>
<td></td>
</tr>
<tr>
<td>lookup</td>
<td></td>
</tr>
</tbody>
</table>
Only becomes an orchestrating command when `local=true`. This forces the `lookup` command to run on the search head and not on any remote peers. A **streaming** (distributable) command when `local=false`, which is the default setting.

<table>
<thead>
<tr>
<th>Command</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>noop</code></td>
<td></td>
</tr>
<tr>
<td><code>re distribute</code></td>
<td></td>
</tr>
</tbody>
</table>

**Dataset processing commands**

A dataset processing command is a command that requires the entire dataset before the command can run. Some of these commands fit into other command types in specific situations or when specific arguments are used.

<table>
<thead>
<tr>
<th>Command</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>anomalousvalue</code></td>
<td>Some modes</td>
</tr>
<tr>
<td><code>anomalydetection</code></td>
<td>Some modes</td>
</tr>
<tr>
<td><code>append</code></td>
<td>Some modes</td>
</tr>
<tr>
<td><code>bin</code></td>
<td>Some modes. A <strong>streaming</strong> command if the <code>span</code> argument is specified.</td>
</tr>
<tr>
<td><code>cluster</code></td>
<td>Some modes</td>
</tr>
<tr>
<td><code>concurrency</code></td>
<td></td>
</tr>
<tr>
<td><code>datamodel</code></td>
<td></td>
</tr>
<tr>
<td><code>dedup</code></td>
<td>Some modes</td>
</tr>
<tr>
<td><code>eventstats</code></td>
<td></td>
</tr>
<tr>
<td><code>fieldsummary</code></td>
<td>Some modes</td>
</tr>
<tr>
<td><code>fillnull</code></td>
<td>Some modes</td>
</tr>
<tr>
<td><code>from</code></td>
<td>Some modes</td>
</tr>
<tr>
<td><code>join</code></td>
<td>Some modes</td>
</tr>
<tr>
<td><code>map</code></td>
<td></td>
</tr>
<tr>
<td><code>outlier</code></td>
<td></td>
</tr>
<tr>
<td><code>pivot</code></td>
<td>Some modes</td>
</tr>
<tr>
<td><code>reverse</code></td>
<td></td>
</tr>
<tr>
<td><code>sort</code></td>
<td></td>
</tr>
</tbody>
</table>
### Splunk SPL for SQL users

This is not a perfect mapping between SQL and Splunk Search Processing Language (SPL), but if you are familiar with SQL, this quick comparison might be helpful as a jump-start into using the search commands.

#### Concepts

The Splunk platform does not store data in a conventional database. Rather, it stores data in a distributed, non-relational, semi-structured database with an implicit time dimension. Relational databases require that all table columns be defined up-front and they do not automatically scale by just plugging in new hardware. However, there are analogues to many of the concepts in the database world.

<table>
<thead>
<tr>
<th>Database Concept</th>
<th>Splunk Concept</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL query</td>
<td>Splunk search</td>
<td>A Splunk search retrieves indexed data and can perform transforming and reporting operations. Results from one search can be &quot;piped&quot;, or transferred, from command to command, to filter, modify, reorder, and group your results.</td>
</tr>
<tr>
<td>table/view</td>
<td>search results</td>
<td>Search results can be thought of as a database view, a dynamically generated table of rows, with columns.</td>
</tr>
<tr>
<td>index</td>
<td>index</td>
<td>All values and fields are indexed by Splunk software, so there is no need to manually add, update, drop, or even think about indexing columns. Everything can be quickly retrieved automatically.</td>
</tr>
</tbody>
</table>
Database Concept | Splunk Concept | Notes
--- | --- | ---
row | result/event | A result in a Splunk search is a list of fields (i.e., column) values, corresponding to a table row. An event is a result that has a timestamp and raw text. Typically an event is a record from a log file, such as:

```
```

column | field | Fields are returned dynamically from a search, meaning that one search might return a set of fields, while another search might return another set. After teaching Splunk software how to extract more fields from the raw underlying data, the same search will return more fields than it previously did. Fields are not tied to a datatype.

database/schema | index/app | A Splunk index is a collection of data, somewhat like a database has a collection of tables. Domain knowledge of that data, how to extract it, what reports to run, etc, are stored in a Splunk application.

### From SQL to Splunk SPL

SQL is designed to search relational database tables which are comprised of columns. SPL is designed to search events, which are comprised of fields. In SQL, you often see examples that use "mytable" and "mycolumn". In SPL, you will see examples that refer to "fields". In these examples, the "source" field is used as a proxy for "table". In Splunk software, "source" is the name of the file, stream, or other input from which a particular piece of data originates, for example `/var/log/messages` or UDP:514.

When translating from any language to another, often the translation is longer because of idioms in the original language. Some of the Splunk search examples shown below could be more concise, but for parallelism and clarity, the SPL table and field names are kept the same as the SQL example.

- SPL searches rarely need the FIELDS command to filter out columns because the user interface provides a more convenient method for filtering. The FIELDS command is used in the SPL examples for
• With SPL, you never have to use the AND operator in Boolean searches, because AND is implied between terms. However when you use the AND or OR operators, they must be specified in uppercase.
• SPL commands do not need to be specified in uppercase. In the these SPL examples, the commands are specified in uppercase for easier identification and clarity.

<table>
<thead>
<tr>
<th>SQL command</th>
<th>SQL example</th>
<th>Splunk SPL example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT *</td>
<td>SELECT * FROM mytable</td>
<td>source=mytable</td>
</tr>
<tr>
<td>WHERE</td>
<td>SELECT * FROM mytable WHERE mycolumn=5</td>
<td>source=mytable mycolumn=5</td>
</tr>
<tr>
<td>SELECT</td>
<td>SELECT mycolumn1, mycolumn2 FROM mytable</td>
<td>source=mytable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FIELDS mycolumn1, mycolumn2</td>
</tr>
<tr>
<td>AND/OR</td>
<td>SELECT * FROM mytable WHERE (mycolumn1=&quot;true&quot;</td>
<td>source=mytable</td>
</tr>
<tr>
<td></td>
<td>OR mycolumn2=&quot;red&quot;) AND mycolumn3=&quot;blue&quot;</td>
<td>AND (mycolumn1=&quot;true&quot; OR mycolumn2=&quot;red&quot;) AND mycolumn3=&quot;blue&quot;</td>
</tr>
<tr>
<td></td>
<td>Note: The AND operator is implied in SPL and does not need to be specified. For this example you could also use:</td>
<td>source=mytable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(mycolumn1=&quot;true&quot; OR mycolumn2=&quot;red&quot;) mycolumn3=&quot;blue&quot;</td>
</tr>
<tr>
<td>AS (alias)</td>
<td>SELECT mycolumn AS column_alias FROM mytable</td>
<td>source=mytable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RENAME mycolumn as column_alias</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FIELDS column_alias</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>SELECT * FROM mytable WHERE mycolumn</td>
<td>source=mytable mycolumn&gt;=1 mycolumn&lt;=5</td>
</tr>
<tr>
<td>SQL command</td>
<td>SQL example</td>
<td>Splunk SPL example</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| BETWEEN 1 AND 5 | | source=metatable  
mycolumn=value  
| STATS avg(mycolumn) BY mycolumn  
| FIELDS mycolumn,  
avg(mycolumn)  
| Several commands use a by-clause to group information, including chart, rare, sort, stats, and timechart. |
| GROUP BY | SELECT mycolumn,  
avg(mycolumn)  
FROM mytable  
WHERE mycolumn=value  
GROUP BY mycolumn  
| source=metatable  
mycolumn=value  
| STATS avg(mycolumn) BY mycolumn  
| FIELDS mycolumn,  
avg(mycolumn)  
| Several commands use a by-clause to group information, including chart, rare, sort, stats, and timechart. |
| HAVING | SELECT mycolumn,  
avg(mycolumn)  
FROM mytable  
WHERE mycolumn=value  
GROUP BY mycolumn  
HAVING avg(mycolumn)=value  
| source=metatable  
mycolumn=value  
| STATS avg(mycolumn) BY mycolumn  
| SEARCH avg(mycolumn)=value  
| FIELDS mycolumn,  
avg(mycolumn) |
| LIKE | SELECT *  
FROM mytable  
WHERE mycolumn LIKE "%some text%"  
| source=metatable  
mycolumn="*some text*"  
| Note: The most common search in Splunk SPL is nearly impossible in SQL - to search all fields for a substring. The following SPL search returns all rows that contain "some text" anywhere:  
| source=metatable "some text" |
| ORDER BY | SELECT *  
FROM mytable  
ORDER BY mycolumn desc  
| source=metatable  
mycolumn=-mycolumn  
| In SPL you use a negative sign (-) in front of a field name to sort in descending order. |
| SELECT DISTINCT | SELECT DISTINCT  
mycolumn1, mycolumn2  
| source=metatable  
<p>| DEDUP mycolumn1 |</p>
<table>
<thead>
<tr>
<th>SQL command</th>
<th>SQL example</th>
<th>Splunk SPL example</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM mytable</td>
<td>FROM mytable</td>
<td>| FIELDS mycolumn1, mycolumn2</td>
</tr>
<tr>
<td>SELECT TOP (5) mycolumn1, mycolumn2 FROM mytable1 WHERE mycolumn3 = &quot;bar&quot; ORDER BY mycolumn1 mycolumn2</td>
<td>Source=mytable1 mycolumn3=&quot;bar&quot;</td>
<td>| FIELDS mycolumn1 mycolumn2 | SORT mycolumn1 mycolumn2 | HEAD 5</td>
</tr>
<tr>
<td>INNER JOIN</td>
<td>SELECT * FROM mytable1 INNER JOIN mytable2 ON mytable1.mycolumn=mytable2.mycolumn</td>
<td>source=mytable1</td>
</tr>
<tr>
<td>Note: There are two other methods to join tables:</td>
<td>Note: There are two other methods to join tables:</td>
<td></td>
</tr>
<tr>
<td>• Use the lookup command to add fields from an external table:</td>
<td>• Use the lookup command to add fields from an external table:</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>| LOOKUP myvaluere lookup mycolumn</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OUTPUT myoutputcolumn</td>
<td></td>
</tr>
<tr>
<td>• Use a subsearch:</td>
<td>• Use a subsearch:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>source=mytable1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[SEARCH source=mytable2 mycolumn2=myvalue | FIELDS mycolumn2]</td>
<td></td>
</tr>
<tr>
<td>If the columns that you want to join on have different names, use the rename command to rename one of the columns. For example, to rename the column in mytable2:</td>
<td>If the columns that you want to join on have different names, use the rename command to rename one of the columns. For example, to rename the column in mytable2:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>source=mytable1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ SEARCH source=mytable2 | RENAME mycolumn2</td>
<td></td>
</tr>
<tr>
<td>SQL command</td>
<td>SQL example</td>
<td>Splunk SPL example</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>AS mycolumn]</td>
<td>To rename the column in mytable1:</td>
</tr>
<tr>
<td></td>
<td>source=mytable1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>JOIN type=inner mycolumn</td>
</tr>
<tr>
<td></td>
<td>You can rename a column regardless of whether you use</td>
<td>You can rename a column regardless of whether you use the JOIN command, a lookup,</td>
</tr>
<tr>
<td></td>
<td>the JOIN command, a lookup, or a subsearch.</td>
<td>or a subsearch.</td>
</tr>
<tr>
<td>LEFT (OUTER) JOIN</td>
<td>SELECT *</td>
<td>source=mytable1</td>
</tr>
<tr>
<td></td>
<td>FROM mytable1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LEFT JOIN mytable2</td>
<td>[SEARCH source=mytable2]</td>
</tr>
<tr>
<td></td>
<td>ON mytable1.mycolumn= mytable2.mycolumn</td>
<td></td>
</tr>
<tr>
<td>SELECT INTO</td>
<td>SELECT *</td>
<td>source=old_mytable</td>
</tr>
<tr>
<td></td>
<td>INTO new_mytable IN mydb2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FROM old_mytable</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> COLLECT is typically used to store expensively calculated fields back</td>
</tr>
<tr>
<td></td>
<td></td>
<td>into your Splunk deployment so that future access is much faster. This current</td>
</tr>
<tr>
<td></td>
<td></td>
<td>example is atypical but shown for comparison to the SQL command. The source will</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be renamed orig_source</td>
</tr>
<tr>
<td>TRUNCATE TABLE</td>
<td>TRUNCATE TABLE mytable</td>
<td>source=mytable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSERT INTO</td>
<td>INSERT INTO mytable VALUES (value1, value2, value3,</td>
<td><strong>Note:</strong> see SELECT INTO. Individual records are not added via the search language,</td>
</tr>
<tr>
<td></td>
<td>....)</td>
<td>but can be added via the API if need be.</td>
</tr>
</tbody>
</table>
### SQL commands

<table>
<thead>
<tr>
<th>SQL command</th>
<th>SQL example</th>
<th>Splunk SPL example</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNION</td>
<td>SELECT mycolumn FROM mytable1 UNION SELECT mycolumn FROM mytable2</td>
<td>source=mytable1</td>
</tr>
<tr>
<td>UNION ALL</td>
<td>SELECT * FROM mytable1 UNION ALL SELECT * FROM mytable2</td>
<td>source=mytable1</td>
</tr>
<tr>
<td>DELETE</td>
<td>DELETE FROM mytable WHERE mycolumn=5</td>
<td>source=mytable1 mycolumn=5</td>
</tr>
<tr>
<td>UPDATE</td>
<td>UPDATE mytable SET column1=value, column2=value,... WHERE some_column=some_value</td>
<td><strong>Note:</strong> There are a few things to think about when updating records in Splunk Enterprise. First, you can just add the new values to your Splunk deployment (see INSERT INTO) and not worry about deleting the old values, because Splunk software always returns the most recent results first. Second, on retrieval, you can always de-duplicate the results to ensure only the latest values are used (see SELECT DISTINCT). Finally, you can actually delete the old records (see DELETE).</td>
</tr>
</tbody>
</table>

**See also**

- Understanding SPL syntax

**SPL data types and clauses**
Data types

bool

The `<bool>` argument value represents the Boolean data type. The documentation specifies 'true' or 'false'. Other variations of Boolean values are accepted in commands. For example, for 'true' you can also use 't', 'T', 'TRUE', or the number one '1'. For 'false', you can use 'f', 'F', 'FALSE', or the number zero '0'.

int

The `<int>` argument value represents the integer data type.

num

The `<num>` argument value represents the number data type.

float

The `<float>` argument value represents the float data type.

Common syntax clauses

bin-span

Syntax: span=(<span-length> | <log-span>)
Description: Sets the size of each bin.
Example: span=2d
Example: span=5m
Example: span=10

by-clause

Syntax: by <field-list>
Description: Fields to group by.
Example: BY addr, port
Example: BY host
**eval-function**

Syntax: abs | case | cidrmatch | coalesce | exact | exp | floor | if | ifnull | isbool | isint | isnotnull | isnull | isnum | isstr | len | like | ln | log | lower | match | max | md5 | min | mvcount | mvindex | mvfilter | now | null | nullif | pi | pow | random | replace | round | searchmatch | sqrt | substr | tostring | trim | ltrim | rtrim | typeof | upper | urldecode | validate

Description: Function used by eval.

Example: md5(field)

Example: typeof(12) + typeof("string") + typeof(1==2) + typeof(badfield)

Example: searchmatch("foo AND bar")

Example: sqrt(9)

Example: round(3.5)

Example: replace(date, "^\d{1,2}/\d{1,2}/", ".\2/\1/"")

Example: pi()

Example: nullif(fielda, fieldb)

Example: random()

Example: pow(x, y)

Example: mvfilter(match(email, ".net") OR match(email, ".org"))

Example: mvindex(multifield, 2)

Example: null()

Example: now()

Example: isbool(field)

Example: exp(3)

Example: floor(1.9)

Example: coalesce(null(), "Returned value", null())

Example: exact(3.14 * num)

Example: case(error == 404, "Not found", error == 500, "Internal Server Error", error == 200, "OK")

Example: cidrmatch("123.132.32.0/25", ip)

Example: abs(number)

Example: isnotnull(field)

Example: substr("string", 1, 3) + substr("string", -3)

Example: if(error == 200, "OK", "Error")

Example: len(field)

Example: log(number, 2)

Example: lower(username)

Example: match(field, "\d{1,3}\d{1,3}\d{1,3}\d{1,3}\$")

Example: max(1, 3, 6, 7, "f\d{1,3}\d{1,3}\d{1,3}\d{1,3}\$\")

Example: like(field, "foo%")

Example: ln(bytes)

Example: mvcount(multifield)
Example:
urldecode("http%3A%2F%2Fwww.splunk.com%2Fdownload%3Fr%3Dheader")

Example: validate(isint(port), "ERROR: Port is not an integer", port >= 1
AND port <= 65535, "ERROR: Port is out of range")

Example: tostring(1==1) + " " + tostring(15, "hex") + " " +
tostring(12345.6789, "commas")

Example: trim(" ZZZZabcZZ ", " Z")

evaluated-field

Syntax: eval(<eval-expression>)
Description: A dynamically evaluated field

field

field-list

regex-expression

Syntax: (^)?<string>(^)?
Description: A Perl Compatible Regular Expression supported by the
PCRE library.
Example: ... | regex _raw="(?<!\d)10.\d{1,3}\d{1,3}\d{1,3}(?!\d)"

single-agg

Syntax: count | stats-func (<field>)
Description: A single aggregation applied to a single field (can be evaluated
field). No wildcards are allowed. The field must be specified, except when
using the special 'count' aggregator that applies to events as a whole.

Example: avg(delay)
Example: sum({date_hour * date_minute})
Example: count

sort-by-clause

Syntax: (\-|\+)<sort-field> ","
Description: List of fields to sort by and their sort order (ascending or
descending)

Example: - time, host
Example: -size, +source
Example: _time, -host
**span-length**

**Syntax:** `<int:span> (<timescale>)?`

**Description:** Span of each bin. If using a timescale, this is used as a time range. If not, this is an absolute bucket "length."

**Example:**
- 2d
- 5m
- 10

**split-by-clause**

**Syntax:** `<field> (<tc-option>) * (<where-clause>)?`

**Description:** Specifies a field to split by. If field is numerical, default discretization is applied.

**stats-agg**

**Syntax:** `<stats-func> (" ( <evaled-field> | <wc-field> )? ")" )?`

**Description:** A specifier formed by an aggregation function applied to a field or set of fields. As of 4.0, it can also be an aggregation function applied to an arbitrary eval expression. The eval expression must be wrapped by "{" and "}". If no field is specified in the parenthesis, the aggregation is applied independently to all fields, and is equivalent to calling a field value of * When a numeric aggregator is applied to a not-completely-numeric field no column is generated for that aggregation.

**Example:**
- count({sourcetype="splunkd"})
- max(size)
- stdev("delay")
- avg(kbps)

**stats-agg-term**

**Syntax:** `<stats-agg> (as <wc-field>)?`

**Description:** A statistical specifier optionally renamed to a new field name.

**Example:**
- count(device) AS numdevices
- avg(kbps)

**subsearch**

**Syntax:** `[<string>]`

**Description:** Specifies a subsearch.

**Example:** [search 404 | select url]
tc-option

**Syntax:** <bins-options> | (usenull=<bool>) | (useother=<bool>) | (nullstr=<string>) |(otherstr=<string>)

**Description:** Options for controlling the behavior of splitting by a field. In addition to the bins-options: usenull controls whether or not a series is created for events that do not contain the split-by field. This series is labeled by the value of the nullstr option, and defaults to NULL. useother specifies if a series should be added for data series not included in the graph because they did not meet the criteria of the <where-clause>. This series is labeled by the value of the otherstr option, and defaults to OTHER.

**Example:** otherstr=OTHERFIELDS
**Example:** usenull=f
**Example:** bins=10

**timeformat**

**Syntax:** timeformat=<string>

**Description:** Set the time format for starttime and endtime terms.

**Example:** timeformat=%m/%d/%Y:%H:%M:%S

**timestamp**

**Syntax:** (MM/DD/YY)?:(HH:MM:SS)?|<int>

**Description:** None

**Example:** 10/1/07:12:34:56
**Example:** -5

**where-clause**

**Syntax:** where <single-agg> <where-comp>

**Description:** Specifies the criteria for including particular data series when a field is given in the tc-by-clause. This optional clause, if omitted, default to "where sum in top10". The aggregation term is applied to each data series and the result of these aggregations is compared to the criteria. The most common use of this option is to select for spikes rather
than overall mass of distribution in series selection. The default value finds the top ten series by area under the curve. Alternately one could replace sum with max to find the series with the ten highest spikes.

**Example:** where max < 10
**Example:** where count notin bottom10
**Example:** where avg > 100
**Example:** where sum in top5

`wc-field`
Evaluation Functions

Evaluation functions

Use the evaluation functions to evaluate an expression, based on your events, and return a result.

Quick reference

See the Supported functions and syntax section for a quick reference list of the evaluation functions.

Commands

You can use evaluation functions with the eval, fieldformat, and where commands, and as part of eval expressions with other commands.

Usage

- All functions that accept strings can accept literal strings or any field.
- All functions that accept numbers can accept literal numbers or any numeric field.

String arguments and fields

For most evaluation functions, when a string argument is expected, you can specify either a literal string or a field name. Literal strings must be enclosed in double quotation marks. In other words, when the function syntax specifies a string you can specify any expression that results in a string. For example, you have a field called name which contains the names of your servers. You want to append the literal string server at the end of the name. You would specify this: name + "server".

Nested functions

You can specify a function as an argument to another function.

In the following example, the cidrmatch function is used as the first argument in the if function.
... | eval isLocal=if(cidrmatch("123.132.32.0/25",ip), "local", "not local")

The following example shows how to use the true() function to provide a default to the case function.

... | eval error=case(status == 200, "OK", status == 404, "Not found", true(), "Other")

**Supported functions and syntax**

There are two ways that you can see information about the supported evaluation functions:

- Function list by category
- Alphabetical list of functions

**Function list by category**

The following table is a quick reference of the supported evaluation functions. This table lists the syntax and provides a brief description for each of the functions. Use the links in the **Type of function** column for more details and examples.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison and Conditional functions</td>
<td>case(X,&quot;Y&quot;,...)</td>
<td>Accepts alternating conditions and values. Returns the first value for which the condition evaluates to TRUE.</td>
</tr>
<tr>
<td></td>
<td>cidrmatch(&quot;X&quot;,Y)</td>
<td>Returns TRUE or FALSE based on whether an IP address matches a CIDR notation.</td>
</tr>
<tr>
<td></td>
<td>coalesce(X,...)</td>
<td>This function takes an arbitrary number of</td>
</tr>
<tr>
<td>Type of function</td>
<td>Supported functions and syntax</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>arguments and returns the first value that is not NULL.</td>
<td>false()</td>
</tr>
<tr>
<td></td>
<td>If the condition X evaluates to TRUE, returns Y, otherwise returns Z.</td>
<td>if(X,Y,Z)</td>
</tr>
<tr>
<td></td>
<td>The function returns TRUE if one of the values in the list matches a value in the field you specify.</td>
<td>in(FIELD, VALUE-LIST)</td>
</tr>
<tr>
<td></td>
<td>Returns TRUE if TEXT matches PATTERN.</td>
<td>like(TEXT, PATTERN)</td>
</tr>
<tr>
<td></td>
<td>Returns TRUE or FALSE based on whether REGEX matches SUBJECT</td>
<td>match(SUBJECT, &quot;REGEX&quot;)</td>
</tr>
<tr>
<td></td>
<td>This function takes no arguments and returns NULL.</td>
<td>null()</td>
</tr>
<tr>
<td></td>
<td>This function is used to compare fields. The function takes two arguments, X and Y, and returns NULL if X = Y. Otherwise it returns X.</td>
<td>nullif(X,Y)</td>
</tr>
<tr>
<td></td>
<td>Use this function to return TRUE if the search string (X) matches the event.</td>
<td>searchmatch(X)</td>
</tr>
<tr>
<td></td>
<td>Returns TRUE.</td>
<td>true()</td>
</tr>
<tr>
<td></td>
<td>validate(X,Y,...)</td>
<td>validate(X,Y,...)</td>
</tr>
</tbody>
</table>
Use this function to return the string Y corresponding to the first expression X that evaluates to FALSE. This function is the opposite of the `case` function.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion functions</td>
<td><code>printf(&quot;format&quot;,arguments)</code></td>
<td>Creates a formatted string based on a format description that you provide.</td>
</tr>
<tr>
<td></td>
<td><code>tonumber(NUMSTR,BASE)</code></td>
<td>Converts a string to a number.</td>
</tr>
<tr>
<td></td>
<td><code>tostring(X,Y)</code></td>
<td>Converts the input, such as a number or a Boolean value, to a string.</td>
</tr>
<tr>
<td>Cryptographic functions</td>
<td><code>md5(X)</code></td>
<td>Computes the md5 hash for the value X.</td>
</tr>
<tr>
<td></td>
<td><code>sha1(X)</code></td>
<td>Computes the sha1 hash for the value X.</td>
</tr>
<tr>
<td></td>
<td><code>sha256(X)</code></td>
<td>Computes the sha256 hash for the value X.</td>
</tr>
<tr>
<td></td>
<td><code>sha512(X)</code></td>
<td>Computes the sha512 hash for the value X.</td>
</tr>
<tr>
<td>Date and Time functions</td>
<td><code>now()</code></td>
<td>Returns the time that the search was started.</td>
</tr>
<tr>
<td></td>
<td><code>relative_time(X,Y)</code></td>
<td>Adjusts the time by a relative time specifier.</td>
</tr>
<tr>
<td></td>
<td><code>strftime(X,Y)</code></td>
<td>Takes a UNIX time and renders it into a human readable format.</td>
</tr>
<tr>
<td></td>
<td><code>strptime(X,Y)</code></td>
<td></td>
</tr>
<tr>
<td>Type of function</td>
<td>Supported functions and syntax</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>time()</td>
<td>Takes a human readable time and renders it into UNIX time.</td>
</tr>
<tr>
<td></td>
<td>isbool(X)</td>
<td>Returns TRUE if the field value is Boolean.</td>
</tr>
<tr>
<td></td>
<td>isint(X)</td>
<td>Returns TRUE if the field value is an integer.</td>
</tr>
<tr>
<td></td>
<td>isnotnull(X)</td>
<td>Returns TRUE if the field value is not NULL.</td>
</tr>
<tr>
<td></td>
<td>isnull(X)</td>
<td>Returns TRUE if the field value is NULL.</td>
</tr>
<tr>
<td></td>
<td>isnum(X)</td>
<td>Returns TRUE if the field value is a number.</td>
</tr>
<tr>
<td></td>
<td>isstr(X)</td>
<td>Returns TRUE if the field value is a string.</td>
</tr>
<tr>
<td></td>
<td>typeof(X)</td>
<td>Returns a string that indicates the field type, such as Number, String, Boolean, and so forth</td>
</tr>
<tr>
<td></td>
<td>abs(X)</td>
<td>Returns the absolute value.</td>
</tr>
<tr>
<td></td>
<td>ceiling(X) or ceil(X)</td>
<td>Rounds the value up to the next highest integer.</td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>exact(X)</td>
<td>Returns the result of a numeric eval calculation with a larger amount of precision in the formatted output.</td>
<td></td>
</tr>
<tr>
<td>exp(X)</td>
<td>Returns the exponential function $e^x$.</td>
<td></td>
</tr>
<tr>
<td>floor(X)</td>
<td>Rounds the value down to the next lowest integer.</td>
<td></td>
</tr>
<tr>
<td>ln(X)</td>
<td>Returns the natural logarithm.</td>
<td></td>
</tr>
<tr>
<td>log(X,Y)</td>
<td>Returns the logarithm of X using Y as the base. If Y is omitted, base 10 is used.</td>
<td></td>
</tr>
<tr>
<td>pi()</td>
<td>Returns the constant pi to 11 digits of precision.</td>
<td></td>
</tr>
<tr>
<td>pow(X,Y)</td>
<td>Returns X to the power of Y, $x^y$.</td>
<td></td>
</tr>
<tr>
<td>round(X,Y)</td>
<td>Returns X rounded to the amount of decimal places specified by Y. The default is to round to an integer.</td>
<td></td>
</tr>
<tr>
<td>sigfig(X)</td>
<td>Rounds X to the appropriate number of significant figures.</td>
<td></td>
</tr>
<tr>
<td>sqrt(X)</td>
<td>Returns the square root of the value.</td>
<td></td>
</tr>
<tr>
<td>commands(X)</td>
<td>Returns a multivalued field that contains a list of the commands</td>
<td></td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>mvappend(X,...)</td>
<td>Returns a multivalue result based on all of values specified.</td>
<td></td>
</tr>
<tr>
<td>mvcount(MVFIELD)</td>
<td>Returns the count of the number of values in the specified field.</td>
<td></td>
</tr>
<tr>
<td>mvdedup(X)</td>
<td>Removes all of the duplicate values from a multivalue field.</td>
<td></td>
</tr>
<tr>
<td>mvfilter(X)</td>
<td>Filters a multivalue field based on an arbitrary Boolean expression X.</td>
<td></td>
</tr>
<tr>
<td>mvfind(MVFIELD,&quot;REGEX&quot;)</td>
<td>Finds the index of a value in a multivalue field that matches the REGEX.</td>
<td></td>
</tr>
<tr>
<td>mvindex(MVFIELD,STARTINDEX,ENDINDEX)</td>
<td>Returns a set of values from a multivalue field described by STARTINDEX and ENDINDEX.</td>
<td></td>
</tr>
<tr>
<td>mvjoin(MVFIELD,STR)</td>
<td>Takes all of the values in a multivalue field and appends them together delimited by STR.</td>
<td></td>
</tr>
<tr>
<td>mvrange(X,Y,Z)</td>
<td>Creates a multivalue field with a range of numbers between X and Y, incrementing by Z.</td>
<td></td>
</tr>
<tr>
<td>mvsort(X)</td>
<td>Returns the values of a multivalue field</td>
<td></td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><code>mvzip(X,Y,&quot;Z&quot;)</code></td>
<td>Takes two multivalue fields, X and Y, and combines them by stitching together the first value of X with the first value of field Y, then the second with the second, and so on.</td>
<td></td>
</tr>
<tr>
<td><code>split(X,&quot;Y&quot;)</code></td>
<td>Returns a mv field splitting X by the delimited character Y.</td>
<td></td>
</tr>
<tr>
<td><code>max(X,...)</code></td>
<td>Returns the maximum of the string or numeric values.</td>
<td></td>
</tr>
<tr>
<td><code>min(X,...)</code></td>
<td>Returns the minimum of the string or numeric values.</td>
<td></td>
</tr>
<tr>
<td><code>random()</code></td>
<td>Returns a pseudo-random integer ranging from zero to $2^{31}-1$.</td>
<td></td>
</tr>
<tr>
<td><code>len(X)</code></td>
<td>Returns the count of the number of characters (not bytes) in the string.</td>
<td></td>
</tr>
<tr>
<td><code>lower(X)</code></td>
<td>Converts the string to lowercase.</td>
<td></td>
</tr>
<tr>
<td><code>ltrim(X,Y)</code></td>
<td>Trims the characters represented in Y from the left side of the string.</td>
<td></td>
</tr>
<tr>
<td><code>replace(X,Y,Z)</code></td>
<td>Returns a string formed by substituting</td>
<td></td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><code>string Z for every occurrence of regex string Y in string X.</code></td>
<td>Returns X with the characters in Y trimmed from the right side.</td>
<td></td>
</tr>
<tr>
<td><code>rtrim(X,Y)</code></td>
<td>Extracts a value from a structured data type (XML or JSON) in X based on a location path in Y.</td>
<td></td>
</tr>
<tr>
<td><code>spath(X,Y)</code></td>
<td>Returns a substring from X based on the starting position Y and the length Z.</td>
<td></td>
</tr>
<tr>
<td><code>substr(X,Y,Z)</code></td>
<td>Trims the characters represented in Y from both sides of the string X.</td>
<td></td>
</tr>
<tr>
<td><code>trim(X,Y)</code></td>
<td>Returns the string in uppercase.</td>
<td></td>
</tr>
<tr>
<td><code>upper(X)</code></td>
<td>Replaces URL escaped characters with the original characters.</td>
<td></td>
</tr>
<tr>
<td><code>urlencode(X)</code></td>
<td>Computes the arc cosine of X.</td>
<td></td>
</tr>
<tr>
<td><code>acos(X)</code></td>
<td>Computes the arc hyperbolic cosine of X.</td>
<td></td>
</tr>
<tr>
<td><code>acosh(X)</code></td>
<td>Computes the arc sine of X.</td>
<td></td>
</tr>
<tr>
<td><code>asin(X)</code></td>
<td>Computes the arc hyperbolic sine of X.</td>
<td></td>
</tr>
<tr>
<td><code>asinh(X)</code></td>
<td>atan(X)</td>
<td></td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td>Type of function</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>atan2(X,Y)</td>
<td>Computes the arc tangent of X, Y.</td>
<td></td>
</tr>
<tr>
<td>atanh(X)</td>
<td>Computes the arc hyperbolic tangent of X.</td>
<td></td>
</tr>
<tr>
<td>cos(X)</td>
<td>Computes the cosine of an angle of X radians.</td>
<td></td>
</tr>
<tr>
<td>cosh(X)</td>
<td>Computes the hyperbolic cosine of X radians.</td>
<td></td>
</tr>
<tr>
<td>hypot(X,Y)</td>
<td>Computes the hypotenuse of a triangle.</td>
<td></td>
</tr>
<tr>
<td>sin(X)</td>
<td>Computes the sine of X.</td>
<td></td>
</tr>
<tr>
<td>sinh(X)</td>
<td>Computes the hyperbolic sine of X.</td>
<td></td>
</tr>
<tr>
<td>tan(X)</td>
<td>Computes the tangent of X.</td>
<td></td>
</tr>
<tr>
<td>tanh(X)</td>
<td>Computes the hyperbolic tangent of X.</td>
<td></td>
</tr>
</tbody>
</table>

**Alphabetical list of functions**

<table>
<thead>
<tr>
<th>Supported functions and syntax</th>
<th>Description</th>
<th>Type of function</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(X)</td>
<td>Returns the absolute value.</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td>acos(X)</td>
<td>Computes the arc cosine of X.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td>Type of function</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>acosh(X)</td>
<td>Computes the arc hyperbolic cosine of X.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>asin(X)</td>
<td>Computes the arc sine of X.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>asinh(X)</td>
<td>Computes the arc hyperbolic sine of X.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>atan(X)</td>
<td>Computes the arc tangent of X.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>atan2(X,Y)</td>
<td>Computes the arc tangent of X,Y.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>atanh(X)</td>
<td>Computes the arc hyperbolic tangent of X.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>case(X,&quot;Y&quot;,...)</td>
<td>Accepts alternating conditions and values. Returns the first value for which the condition evaluates to TRUE.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td>cidrmatch(&quot;X&quot;,Y)</td>
<td>Returns TRUE or FALSE based on whether an IP address matches a CIDR notation.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td>ceiling(X)</td>
<td>Rounds the value up</td>
<td>Mathematical</td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td>Type of function</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>coalesce(X,...)</td>
<td>This function takes an arbitrary number of arguments and returns the first value that is not NULL.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td>commands(X)</td>
<td>Returns a multivalued field that contains a list of the commands used in X.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td>cos(X)</td>
<td>Computes the cosine of an angle of X radians.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>cosh(X)</td>
<td>Computes the hyperbolic cosine of X radians.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>exact(X)</td>
<td>Returns the result of a numeric eval calculation with a larger amount of precision in the formatted output.</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td>exp(X)</td>
<td>Returns the exponential function $e^x$.</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td>false()</td>
<td>Returns FALSE.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td>floor(X)</td>
<td>Rounds the value down to the next lowest integer.</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td>hypot(X,Y)</td>
<td>Computes the</td>
<td>Trigonometry</td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td>Type of function</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>hypotenuse of a triangle.</td>
<td></td>
<td>and Hyperbolic functions</td>
</tr>
<tr>
<td>if(X,Y,Z)</td>
<td>If the condition X evaluates to TRUE, returns Y, otherwise returns Z.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td>in(VALUE-LIST)</td>
<td>The function returns TRUE if one of the values in the list matches a value in the field you specify.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td>isbool(X)</td>
<td>Returns TRUE if the field value is Boolean.</td>
<td>Informational functions</td>
</tr>
<tr>
<td>isint(X)</td>
<td>Returns TRUE if the field value is an integer.</td>
<td>Informational functions</td>
</tr>
<tr>
<td>isnonnull(X)</td>
<td>Returns TRUE if the field value is not NULL.</td>
<td>Informational functions</td>
</tr>
<tr>
<td>isnull(X)</td>
<td>Returns TRUE if the field value is NULL.</td>
<td>Informational functions</td>
</tr>
<tr>
<td>isnum(X)</td>
<td>Returns TRUE if the field value is a number.</td>
<td>Informational functions</td>
</tr>
<tr>
<td>isstr(X)</td>
<td>Returns TRUE if the field value is a string.</td>
<td>Informational functions</td>
</tr>
<tr>
<td>len(X)</td>
<td>Returns the count of the number of characters (not bytes) in the string.</td>
<td>Text functions</td>
</tr>
<tr>
<td>like(TEXT, PATTERN)</td>
<td>Returns TRUE if TEXT matches PATTERN.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td>ln(X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td>Type of function</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td><strong>log(X,Y)</strong></td>
<td>Returns the logarithm of X using Y as the base. If Y is omitted, base 10 is used.</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td><strong>lower(X)</strong></td>
<td>Converts the string to lowercase.</td>
<td>Text functions</td>
</tr>
<tr>
<td><strong>ltrim(X,Y)</strong></td>
<td>Trims the characters represented in Y from the left side of the string.</td>
<td>Text functions</td>
</tr>
<tr>
<td><strong>match(SUBJECT, &quot;REGEX&quot;)</strong></td>
<td>Returns TRUE or FALSE based on whether REGEX matches SUBJECT.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td><strong>max(X,...)</strong></td>
<td>Returns the maximum of the string or numeric values.</td>
<td>Statistical eval functions</td>
</tr>
<tr>
<td><strong>md5(X)</strong></td>
<td>Computes the md5 hash for the value X.</td>
<td>Cryptographic functions</td>
</tr>
<tr>
<td><strong>min(X,...)</strong></td>
<td>Returns the minimum of the string or numeric values.</td>
<td>Statistical eval functions</td>
</tr>
<tr>
<td><strong>mvappend(X,...)</strong></td>
<td>Returns a multivalue result based on all of values specified.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td><strong>mvcount(MVFIELD)</strong></td>
<td>Returns the count of the number of values in the specified field.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td><strong>mvdedup(X)</strong></td>
<td>Removes all of the duplicate values from a multivalue field.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td><strong>mvfilter(X)</strong></td>
<td>Filters a multivalue field based on an arbitrary Boolean</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td>Type of function</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>mvfind(MVFIELD,&quot;REGEX&quot;)</td>
<td>Finds the index of a value in a multivalue field that matches the REGEX.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td>mvindex(MVFIELD,STARTINDEX,ENDINDEX)</td>
<td>Returns a set of values from a multivalue field described by STARTINDEX and ENDINDEX.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td>mvjoin(MVFIELD,STR)</td>
<td>Takes all of the values in a multivalue field and appends them together delimited by STR.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td>mvrange(X,Y,Z)</td>
<td>Creates a multivalue field with a range of numbers between X and Y, incrementing by Z.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td>mvsort(X)</td>
<td>Returns the values of a multivalue field sorted lexicographically.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td>mvzip(X,Y,&quot;Z&quot;)</td>
<td>Takes two multivalue fields, X and Y, and combines them by stitching together the first value of X with the first value of field Y, then the second with the second, and so on.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td>now()</td>
<td>Returns the time that the search was started.</td>
<td>Date and Time functions</td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td>Type of function</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>null()</td>
<td>This function takes no arguments and returns NULL.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td>nullif(X,Y)</td>
<td>This function is used to compare fields. The function takes two arguments, X and Y, and returns NULL if X = Y. Otherwise it returns X.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td>pi()</td>
<td>Returns the constant pi to 11 digits of precision.</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td>pow(X,Y)</td>
<td>Returns X to the power of Y, $X^Y$.</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td>printf(&quot;format&quot;,arguments)</td>
<td>Creates a formatted string based on a format description that you provide.</td>
<td>Conversion functions</td>
</tr>
<tr>
<td>random()</td>
<td>Returns a pseudo-random integer ranging from zero to $2^{31}-1$.</td>
<td>Statistical eval functions</td>
</tr>
<tr>
<td>relative_time(X,Y)</td>
<td>Adjusts the time by a relative time specifier.</td>
<td>Date and Time functions</td>
</tr>
<tr>
<td>replace(X,Y,Z)</td>
<td>Returns a string formed by substituting string Z for every occurrence of regex string Y in string X.</td>
<td>Text functions</td>
</tr>
<tr>
<td>round(X,Y)</td>
<td>Returns X rounded to the amount of decimal places specified by Y. The default is to round to an integer.</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td>Type of function</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>rtrim(X,Y)</td>
<td>Returns X with the characters in Y trimmed from the right side.</td>
<td>Text functions</td>
</tr>
<tr>
<td>searchmatch(X)</td>
<td>Use this function to return TRUE if the search string (X) matches the event.</td>
<td>Comparison and Conditional functions</td>
</tr>
<tr>
<td>sha1(X)</td>
<td>Computes the md5 hash for the value X.</td>
<td>Cryptographic functions</td>
</tr>
<tr>
<td>sha256(X)</td>
<td>Computes the md5 hash for the value X.</td>
<td>Cryptographic functions</td>
</tr>
<tr>
<td>sha512(X)</td>
<td>Computes the sha512 hash for the value X.</td>
<td>Cryptographic functions</td>
</tr>
<tr>
<td>sigfig(X)</td>
<td>Rounds X to the appropriate number of significant figures.</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td>sin(X)</td>
<td>Computes the sine of X.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>sinh(X)</td>
<td>Computes the hyperbolic sine of X.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td>spath(X,Y)</td>
<td>Extracts a value from a structured data type (XML or JSON) in X based on a location path in Y.</td>
<td>Text functions</td>
</tr>
<tr>
<td>split(X,&quot;Y&quot;)</td>
<td>Returns a mv field splitting X by the delimited character Y.</td>
<td>Multivalue eval functions</td>
</tr>
<tr>
<td>sqrt(X)</td>
<td>Returns the square root of the value.</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td>Type of function</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td><code>strftime(X,Y)</code></td>
<td>Takes a UNIX time and renders it into a human readable format.</td>
<td>Date and Time functions</td>
</tr>
<tr>
<td><code>strptime(X,Y)</code></td>
<td>Takes a human readable time and renders it into UNIX time.</td>
<td>Date and Time functions</td>
</tr>
<tr>
<td><code>substr(X,Y,Z)</code></td>
<td>Returns a substring from X based on the starting position Y and the length Z.</td>
<td>Text functions</td>
</tr>
<tr>
<td><code>tan(X)</code></td>
<td>Computes the tangent of X.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td><code>tanh(X)</code></td>
<td>Computes the hyperbolic tangent of X.</td>
<td>Trigonometry and Hyperbolic functions</td>
</tr>
<tr>
<td><code>time()</code></td>
<td>The time that eval function was computed. The time will be different for each event, based on when the event was processed.</td>
<td>Date and Time functions</td>
</tr>
<tr>
<td><code>tonumber(NUMSTR,BASE)</code></td>
<td>Converts a string to a number.</td>
<td>Conversion functions</td>
</tr>
<tr>
<td><code>tostring(X,Y)</code></td>
<td>Converts the input, such as a number or a Boolean value, to a string.</td>
<td>Conversion functions</td>
</tr>
<tr>
<td><code>trim(X,Y)</code></td>
<td>Trims the characters represented in Y from both sides of the string X.</td>
<td>Text functions</td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td>Type of function</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>true()</td>
<td>Returns TRUE.</td>
<td></td>
</tr>
<tr>
<td>typeof(X)</td>
<td>Returns a string that indicates the field type, such as Number, String, Boolean, and so forth.</td>
<td>Informational functions</td>
</tr>
<tr>
<td>upper(X)</td>
<td>Returns the string in uppercase.</td>
<td>Text functions</td>
</tr>
<tr>
<td>urldecode(X)</td>
<td>Replaces URL escaped characters with the original characters.</td>
<td>Text functions</td>
</tr>
<tr>
<td>validate(X,Y,...)</td>
<td>Use this function to return the string Y corresponding to the first expression X that evaluates to FALSE. This function is the opposite of the case function.</td>
<td>Comparison and Conditional functions</td>
</tr>
</tbody>
</table>

**See also**

**Topics:**
Statistical and charting functions

**Commands:**
eval
eval fieldformat
where

**Splunk Answers**

Have questions? Visit Splunk Answers and search for a specific function or command.
Comparison and Conditional functions

The following list contains the functions that you can use to compare values or specify conditional statements.

For information about using string and numeric fields in functions, and nesting functions, see Evaluation functions.

case(X,"Y",...)

Description

Accepts alternating conditions and values. Returns the first value for which the condition evaluates to TRUE.

This function takes pairs of arguments X and Y. The X arguments are Boolean expressions that are evaluated from first to last. When the first X expression is encountered that evaluates to TRUE, the corresponding Y argument is returned. The function defaults to NULL if none are true.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic example

This example uses the sample data from the Search Tutorial, but should work with any format of Apache Web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

The following example returns descriptions for the corresponding http status code.

sourcetype=access_* | eval description=case(status == 200, "OK", status ==404, "Not found", status == 500, "Internal Server Error") | table status description

The results appear on the Statistics tab and look something like this:
For an example of how to display a default value when that status does not match one of the values specified, see the *True function*.

**Extended example**

This example shows you how to use the `case` function in two different ways, to create categories and to create a custom sort order.

This example uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), and so forth, for each earthquake recorded.

You can download a current CSV file from the **USGS Earthquake Feeds** and upload the file to your Splunk instance if you want follow along with this example.

You want classify earthquakes based on depth. Shallow-focus earthquakes occur at depths less than 70 km. Mid-focus earthquakes occur at depths between 70 and 300 km. Deep-focus earthquakes occur at depths greater than 300 km. We'll use Low, Mid, and Deep for the category names.

```
source=all_month.csv | eval Description=case(depth<=70, "Low", depth>70 AND depth<=300, "Mid", depth>300, "Deep") | stats count min(mag) max(mag) by Description
```

The `eval` command is used to create a field called `Description`, which takes the value of "Low", "Mid", or "Deep" based on the Depth of the earthquake. The `case()` function is used to specify which ranges of the depth fits each description. For example, if the depth is less than 70 km, the earthquake is characterized as a shallow-focus quake; and the resulting `Description` is Low.
The search also pipes the results of the `eval` command into the `stats` command to count the number of earthquakes and display the minimum and maximum magnitudes for each Description.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>Description</th>
<th>count</th>
<th>min(Mag)</th>
<th>max(Mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep</td>
<td>35</td>
<td>4.1</td>
<td>6.7</td>
</tr>
<tr>
<td>Low</td>
<td>6236</td>
<td>-0.60</td>
<td>7.70</td>
</tr>
<tr>
<td>Mid</td>
<td>635</td>
<td>0.8</td>
<td>6.3</td>
</tr>
</tbody>
</table>

You can sort the results in the Description column by clicking the sort icon in Splunk Web. However in this example the order would be alphabetical returning results in Deep, Low, Mid or Mid, Low, Deep order.

You can also use the `case` function to sort the results in a custom order, such as Low, Mid, Deep. You create the custom sort order by giving the values a numerical ranking and then sorting based on that ranking.

```
source=all_month.csv | eval Description=case(depth<=70, "Low", depth>70 AND depth<=300, "Mid", depth>300, "Deep") | stats count min(mag) max(mag) by Description | eval sort_field=case(Description="Low", 1, Description="Mid", 2, Description="Deep",3) | sort sort_field
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>Description</th>
<th>count</th>
<th>min(Mag)</th>
<th>max(Mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>6236</td>
<td>-0.60</td>
<td>7.70</td>
</tr>
<tr>
<td>Mid</td>
<td>635</td>
<td>0.8</td>
<td>6.3</td>
</tr>
<tr>
<td>Deep</td>
<td>35</td>
<td>4.1</td>
<td>6.7</td>
</tr>
</tbody>
</table>

`cidrmatch("X",Y)`

**Description**

Returns TRUE or FALSE based on whether an IP address matches a CIDR notation.

Use this function to determine if an IP address belongs to a particular subnet. This function returns TRUE, when IP address Y belongs to a particular subnet X. Both X and Y are string arguments. X is the CIDR subnet. Y is the IP address to match with the subnet. This function is compatible with IPv6.
**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

The following example uses the `cidrmatch` and `if` functions to set a field, `isLocal`, to "local" if the field `ip` matches the subnet. If the `ip` field does not match the subnet, the `isLocal` field is set to "not local".

```
... | eval isLocal=if(cidrmatch("123.132.32.0/25",ip), "local", "not local")
```

The following example uses the `cidrmatch` function as a filter to remove events that do not match the `ip` address:

```
... | where cidrmatch("123.132.32.0/25", ip)
```

**coalesce(X,...)**

**Description**

This function takes an arbitrary number of arguments and returns the first value that is not NULL.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

You have a set of events where the IP address is extracted to either `clientip` or `ipaddress`. This example defines a new field called `ip`, that takes the value of either the `clientip` field or `ipaddress` field, depending on which field is not NULL (does not exist in that event). If both the `clientip` and `ipaddress` field exist in the event, this function returns the first argument, the `clientip` field.

```
... | eval ip=coalesce(clientip,ipaddress)
```
false()

Description

Use this function to return FALSE.

This function enables you to specify a conditional that is obviously false, for example 1==0. You do not specify a field with this function.

Usage

This function is often used as an argument with other functions.

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic examples

if(X,Y,Z)

Description

If the condition X evaluates to TRUE, returns Y, otherwise returns Z.

This function takes three arguments. The first argument X must be a Boolean expression. If X evaluates to TRUE, the result is the second argument Y. If X evaluates to FALSE, the result evaluates to the third argument Z.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

The if function is frequently used with other functions. See Basic examples.

Basic examples

The following example looks at the values of the field error. If error=200, the function returns err=OK. Otherwise the function returns err=Error.

... | eval err=if(error == 200, "OK", "Error")
The following example uses the `cidrmatch` and `if` functions to set a field, `isLocal`, to "local" if the field `ip` matches the subnet. If the `ip` field does not match the subnet, the `isLocal` field is set to "not local".

```plaintext
... | eval isLocal=if(cidrmatch("123.132.32.0/25",ip), "local", "not local")
```

### `in(FIELD, VALUE-LIST)`

**Description**

The function returns TRUE if one of the values in the list matches a value in the field you specify.

This function takes a list of comma-separated values.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions with other commands.

The following syntax is supported:

```plaintext
...| where in(field,"value1","value2", ...)
...| where field in("value1","value2", ...)
...| eval new_field=if(in(field,"value1","value2", ...), "value-if_true","value-if-false")
```

The `eval` command cannot accept a Boolean value. You must specify the `IN` function inside the `IF` function, which can accept a Boolean value as input.

The string values must be enclosed in quotation marks. You cannot specify wildcard characters with the values to specify a group of similar values, such as HTTP error codes or CIDR IP address ranges. Use the `IN` operator instead.

The `IN` operator is similar to the `in` function. You can use the `IN` operator with the `search` and `tstats` commands. You can use wildcard characters in the `VALUE-LIST` with these commands.

**Basic examples**

The following example uses the `where` command to return `in=TRUE` if one of the values in the `status` field matches one of the values in the list.
The following example uses the `in` function as the first parameter for the `if` function. The evaluation expression returns TRUE if the value in the `status` field matches one of the values in the list.

```plaintext
... | eval error=if(in(status, "error", "failure", "severe"), "true", "false")
```

**Extended example**

The following example combines the `in` function with the `if` function to evaluate the `status` field. The value of `true` is placed in the new field `error` if the `status` field contains one of the values 404, 500, or 503. Then a count is performed of the values in the `error` field.

```plaintext
... | eval error=if(in(status, "404","500","503"), "true", "false") | stats count by error
```

**See also**

Blogs
  Smooth operator | Searching for multiple field values

**`like(TEXT, PATTERN)`**

**Description**

This function returns TRUE if `TEXT` matches `PATTERN`.

This function takes two arguments, a string to match `TEXT` and a string expression to match `PATTERN`. It returns TRUE if, and only if, `TEXT` matches `PATTERN`. The pattern language supports an exact text match, as well as percent ( % ) characters for wildcards and underscore ( _ ) characters for a single character match.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.
**Basic examples**

The following example returns `like=TRUE` if the field value starts with `foo`:

```
... | eval is_a_foo=if(like(field, "foo"%), "yes a foo", "not a foo")
```

The following example uses the `where` command to return `like=TRUE` if the field value starts with `foo`:

```
... | where like(field, "foo")
```

**match(SUBJECT, "REGEX")**

**Description**

This function returns TRUE or FALSE based on whether REGEX matches SUBJECT.

This function compares the regex string `REGEX` to the value of `SUBJECT` and returns a Boolean value. It returns TRUE if the REGEX can find a match against any substring of `SUBJECT`.

**Usage**

The `match` function is regex based. For example use the backslash (\) character to escape a special character, such as a quotation mark. Use the pipe (|) character to specify an OR condition.

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

The following example returns TRUE if, and only if, `field` matches the basic pattern of an IP address. This examples uses the caret (^) character and the dollar ($) symbol to perform a full match.

```
... | eval n=if(match(field, "^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\$"), 1, 0)
```
The following example uses the `match` function in an `<eval-expression>`. The SUBJECT is a calculated field called `test`. The "REGEX" is the string `yes`.

This example uses the `match` function in an `<eval-expression>`. The SUBJECT is a calculated field called `test`. The "REGEX" is the string `yes`.

```
... | eval matches = if(match(test,"yes"), 1, 0)
```

If the value is stored with quotation marks, you must use the backslash (`\`) character to escape the embedded quotation marks. For example:

```
| makeresults | eval test="\"yes\"" | eval matches = if(match(test, "\"yes\""), 1, 0)
```

**null()**

**Description**

This function takes no arguments and returns NULL. The evaluation engine uses NULL to represent "no value". Setting a field value to NULL clears the field value.

**Usage**

NULL values are field values that are missing in a some results but present in another results.

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

Suppose you want to calculate the average of the values in a field, but several of the values are zero. If the zeros are placeholders for no value, the zeros will interfere with creating an accurate average. You can use the `null` function to remove the zeros.

**See also**

- You can use the `fillnull` command to replace NULL values with a specified value.
- You can use the `nullif(X,Y)` function to compare two fields and return NULL if X = Y.
nullif(X,Y)

Description

This function is used to compare fields. The function takes two arguments, X and Y, and returns NULL if X = Y. Otherwise it returns X.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic examples

The following example returns NULL if fieldA=fieldB. Otherwise the function returns fieldA.

... | eval n=nullif(fieldA,fieldB)

searchmatch(X)

Description

Use this function to return TRUE if the search string (X) matches the event.

This function takes one argument X, which is a search string. The function returns TRUE if, and only if, the event matches the search string.

Usage

You must use the searchmatch function inside an if function.

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic examples

The following example uses a pipe ( | ) character to specify an OR condition in the searchmatch function.
true()

**Description**

Use this function to return TRUE.

This function enables you to specify a condition that is obviously true, for example $1==1$. You do not specify a field with this function.

**Usage**

This function is often used as an argument with other functions.

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of `eval` expressions.

**Basic examples**

This example uses the sample data from the Search Tutorial, but should work with any format of Apache Web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to [get the tutorial data into Splunk](https://www.splunk.com). Use the time range **Yesterday** when you run the search.

The following example shows how to use the `true()` function to provide a default value to the `case` function. If the values in the status field are not 200, or 404, the value used is Other.

```
sourcetype=access_* | eval description=case(status==200,"OK", status==404, "Not found", true(), "Other") | table status description
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>status</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>408</td>
<td>Other</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
</tbody>
</table>
### validate(X,Y,...)

**Description**

Use this function to return the string Y corresponding to the first expression X that evaluates to FALSE. This function is the opposite of the `case` function.

This function takes pairs of arguments, Boolean expressions X and strings Y. The function returns the string Y corresponding to the first expression X that evaluates to FALSE. This function defaults to NULL if all evaluate to TRUE.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

The following example runs a simple check for valid ports.

```plaintext
... | eval n=validate(isint(port), "ERROR: Port is not an integer", port >= 1 AND port <= 65535, "ERROR: Port is out of range")
```

**Conversion functions**

The following list contains the functions that you can use to convert numbers to strings and strings to numbers.

For information about using string and numeric fields in functions, and nesting functions, see [Evaluation functions](#evaluation-functions).
printf("format",arguments)

**Description**

The `printf` function builds a string value, based on the a string format and the arguments that you specify.

- You can specify zero or more arguments. The arguments can be string values, numbers, computations, or fields.

The SPL `printf` function is similar to the C `sprintf()` function and similar functions in other languages such as Python, Perl, and Ruby.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**format**

**Description:** The `format` is a character string that can include one or more format conversion specifiers. Each conversion specifier can include optional components such as flag characters, width specifications, and precision specifications. The `format` must be enclosed in quotation marks.

**Syntax:** "(%[flags][width][.precision]<conversionspecifier>)..."

**arguments**

**Description:** The `arguments` are optional and can include the width, precision, and the value to format. The value can be a string, number, or field name.

**Syntax:** `[width][.precision][value]`

**Supported conversion specifiers**

The following table describes the supported conversion specifiers.

<table>
<thead>
<tr>
<th>Conversion specifier</th>
<th>Alias</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a or %A</td>
<td></td>
<td>Floating point number in hexadecimal format</td>
<td>This example returns the value of <code>pi</code> to 3 decimal points, in hexadecimal format.</td>
</tr>
<tr>
<td>Conversion specifier</td>
<td>Alias</td>
<td>Description</td>
<td>Examples</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>printf(&quot;%.3A&quot;,pi())</code> which returns 0X1.922P+1</td>
</tr>
<tr>
<td>%c</td>
<td></td>
<td>Single Unicode code point</td>
<td>This example returns the unicode code point for 65 and the first letter of the string &quot;Foo&quot;. <code>printf(&quot;%c,%c&quot;,65,&quot;Foo&quot;)</code> which returns A,F</td>
</tr>
<tr>
<td>%d</td>
<td>%i</td>
<td>Signed decimal integer</td>
<td>This example returns the positive or negative integer values, including any signs specified with those values. <code>printf(&quot;%d,%i,%d&quot;,-2,+4,30)</code> which returns -2,4,30</td>
</tr>
<tr>
<td>%e or %E</td>
<td></td>
<td>Floating point number, exponential format</td>
<td>This example returns the number 5139 in exponential format with 2 decimal points. <code>printf(&quot;%.2e&quot;,5139)</code> which returns 5.14e+03</td>
</tr>
<tr>
<td>%f or %F</td>
<td></td>
<td>Floating point number</td>
<td>This example returns the value of <code>pi</code> to 2 decimal points. <code>printf(&quot;%.2f&quot;,pi())</code> which returns 3.14</td>
</tr>
<tr>
<td>%g or %G</td>
<td></td>
<td>Floating point number. This specifier uses either %e or %f depending on the range of the numbers being formatted.</td>
<td>This example returns the value of <code>pi</code> to 2 decimal points (using the %f specifier) and the number 123 in exponential format with 2 decimal points (using %e specifier). <code>printf(&quot;%.2g,%.2g&quot;,pi(),123)</code> which returns 3.1,1.2e+02</td>
</tr>
<tr>
<td>%o</td>
<td></td>
<td>Unsigned octal number</td>
<td>This example returns the base-8 number for 255.</td>
</tr>
</tbody>
</table>

79
<table>
<thead>
<tr>
<th>Conversion specifier</th>
<th>Alias</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>%s</td>
<td>%z</td>
<td>String</td>
<td><code>printf(&quot;%s\&quot;, &quot;foo\&quot;, &quot;bar\&quot;)</code> which returns <code>foobar</code></td>
</tr>
<tr>
<td>%u</td>
<td></td>
<td>Unsigned, or non-negative, decimal integer</td>
<td><code>printf(&quot;%u\&quot;, 99)</code> which returns <code>99</code></td>
</tr>
<tr>
<td>%x or %X</td>
<td>%p</td>
<td>Unsigned hexadecimal number (lowercase or uppercase)</td>
<td><code>printf(&quot;%x,%X,%p\&quot;, 10, 10, 10)</code> which returns <code>a,A,A</code></td>
</tr>
<tr>
<td>%%%</td>
<td></td>
<td>Percent sign</td>
<td><code>printf(&quot;100\%\&quot;)</code> which returns <code>100%</code></td>
</tr>
</tbody>
</table>

**Flag characters**

The following table describes the supported flag characters.

<table>
<thead>
<tr>
<th>Flag characters</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>single quote or apostrophe ('), %d</td>
<td>Adds commas as the thousands separator.</td>
<td><code>printf(&quot;%d\&quot;, 12345)</code> which returns <code>12,345</code></td>
</tr>
<tr>
<td>Flag characters</td>
<td>Description</td>
<td>Examples</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dash or minus ( - )</td>
<td>Left justify. If this flag is not specified, the is right-justified.</td>
<td>which returns 1</td>
</tr>
<tr>
<td>zero ( 0 )</td>
<td>Zero pad</td>
<td>This example returns the value in the argument with leading zeros such that the number has 4 digits.</td>
</tr>
<tr>
<td>plus ( + )</td>
<td>Always include the sign (+ or -). If this flag is not specified, the conversion displays a sign only for negative values.</td>
<td>printf(&quot;%+4d&quot;,1) which returns +1</td>
</tr>
<tr>
<td>&lt;space&gt;</td>
<td>Reserve space for the sign. If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a &lt;space&gt; is added as a prefixed to the result. If both the &lt;space&gt; and + flags are specified, the &lt;space&gt; flag is ignored.</td>
<td>printf(&quot;%-4d&quot;,1) which returns 1</td>
</tr>
<tr>
<td>hash, number, or pound ( # )</td>
<td>Use an alternate form. For the % conversion specifier, the # flag increases the precision to force the first digit of the result to be zero. For %x or %X conversion specifiers, a non-zero result has 0x (or 0X) prefixed to it. For %a, %A, %e, %E, %f, %F, %g, and G conversion specifiers, the result always contains a radix character, even if no digits follow the radix character. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For %g and %G conversion specifiers, trailing zeros are not removed from the result as they normally are. For other conversion specifiers, the behavior is undefined.</td>
<td>printf(&quot;%#x&quot;, 1) which returns 0x1</td>
</tr>
</tbody>
</table>
Specifying field width

You can use an asterisk ( * ) with the `printf` function to return the field width or precision from an argument.

Examples
The following example returns the positive or negative integer values, including any signs specified with those values.

```c
printf("%*d", 5, 123) which returns 123
```

The following example returns the floating point number with 1 decimal point.

```c
printf("%.1f", 1, 1.23) which returns 1.2
```

The following example returns the value of `pi()` in exponential format with 2 decimal points.

```c
printf("%.2e", 9, 2, pi()) which returns 3.14e+00
```

The field width can be expressed using a number or an argument denoted with an asterisk ( * ) character.

<table>
<thead>
<tr>
<th>Field width specifier</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>The minimum number of characters to print. If the value to print is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.</td>
<td></td>
</tr>
<tr>
<td>* (asterisk)</td>
<td>The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.</td>
<td></td>
</tr>
</tbody>
</table>
### Specifying precision

<table>
<thead>
<tr>
<th>Precision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d, %i, %o, %u, %x or %X</td>
<td>Precision specifies the minimum number of digits to be return. If the value to be return is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is returned for the value 0.</td>
</tr>
<tr>
<td>%a or %A, %e or %E, %f or %F</td>
<td>This is the number of digits to be returned after the decimal point. The default is 6.</td>
</tr>
<tr>
<td>%g or %G</td>
<td>This is the maximum number of significant digits to be returned.</td>
</tr>
<tr>
<td>%s</td>
<td>This is the maximum number of characters to be returned. By default all characters are printed until the ending null character is encountered.</td>
</tr>
</tbody>
</table>

### Unsupported conversion specifiers

There are a few conversion specifiers from the C `sprintf()` function that are not supported, including:

- %C, however %c is supported
- %n
- %S, however %s is supported
- %<num>$ specifier for picking which argument to use

### Basic examples

This example creates a new field called `new_field` and creates string values based on the values in `field_one` and `field_two`. The values are formatted with 4 digits before the decimal and 4 digits after the decimal. The `-` specifies to left justify the string values. The `30` specifies the width of the field.

```bash
...| eval new_field=printf("%04.4f %-30s",field_one,field_two)
```
tonumber(NUMSTR,BASE)

Description

This function converts the input string NUMSTR to a number. NUMSTR can be a field name or a value.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

BASE is optional and used to define the base of the number in NUMSTR. BASE can be 2 to 36. The default is 10 to correspond to the decimal system.

If the tonumber function cannot parse a field value to a number, for example if the value contains a leading and trailing space, the function returns NULL. Use the trim function to remove leading or trailing spaces.

If the tonumber function cannot parse a literal string to a number, it returns an error.

Basic examples

The following example converts the string values for the store_sales field to numbers.

... | eval n=tonumber(store_sales)

The following example takes the hexadecimal number and uses a BASE of 16 to return the number "164".

... | eval n=tonumber("0A4",16)

The following example trims any leading or trailing spaces from the values in the celsius field before converting it to a number.

... | eval temperature=tonumber(trim(celsius))
tostring(X,Y)

Description

This function converts the input value to a string. If the input value is a number, it reformats it as a string. If the input value is a Boolean value, it returns the corresponding string value, "True" or "False".

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

This function requires at least one argument X.

When used with the eval command, the values might not sort as expected because the values are converted to ASCII. Use the fieldformat command with the tostring function to format the displayed values. The underlying values are not changed with the fieldformat command.

If X is a number, the second argument Y is optional and can be "hex", "commas", or "duration".

<table>
<thead>
<tr>
<th>Examples</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tostring(X,&quot;hex&quot;)</td>
<td>Converts X to hexadecimal.</td>
</tr>
<tr>
<td>tostring(X,&quot;commas&quot;)</td>
<td>Formats X with commas. If the number includes decimals, the function rounds to nearest two decimal places.</td>
</tr>
<tr>
<td>tostring(X,&quot;duration&quot;)</td>
<td>Converts seconds X to the readable time format HH:MM:SS.</td>
</tr>
</tbody>
</table>

Basic examples

The following example returns "True 0xF 12,345.68".

```plaintext
... | eval n=tostring(1==1) + " " + tostring(15, "hex") + " " + tostring(12345.6789, "commas")
```

The following example returns foo=615 and foo2=00:10:15. The 615 seconds is converted into minutes and seconds.
... | eval foo=615 | eval foo2 = tostring(foo, "duration")

The following example formats the column totalSales to display values with a currency symbol and commas. You must use a period between the currency value and the `tostring` function.

... | fieldformat totalSales="$".tostring(totalSales,"comma")

See also

Commands
- `convert`

Functions
- `strftime`

Cryptographic functions

The following list contains the functions that you can use to compute the secure hash of string values.

For information about using string and numeric fields in functions, and nesting functions, see Evaluation functions.

`md5(X)`

*Description*

This function computes and returns the MD5 hash of a string value X.

*Usage*

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

*Basic examples*

The following example returns a new field n with a message-digest (MD5) 128-bit hash value.

... | eval n=md5(field)
The following example creates a large random string.

| makeresults count=32768 | eval message=md5("". random()) | stats values(message) as message | eval message = mvjoin(message, "")

- The `makeresults` command creates 32768 results with timestamps.
- The `eval` command creates a new field called `message`:
  - The `random` function returns a random numeric field value for each of the 32768 results. The "". make the numeric number generated by the `random` function into a string value.
  - The `md5` function creates a 128-bit hash value from the string value.
  - The results of the `md5` function are placed into the `message` field created by the `eval` command.
- The `stats` command with the `values` function is used to convert the individual random values into one multivalue result.
- The `eval` command with the `mvjoin` function is used to combine the multivalue entry into a single value.

**sha1(X)**

_**Description**_

This function computes and returns the secure hash of a string value X based on the FIPS compliant SHA-1 hash function.

_**Usage**_

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

_**Basic example**_

```plaintext
... | eval n=sha1(field)
```

**sha256(X)**

_**Description**_

This function computes and returns the secure hash of a string value X based on the FIPS compliant SHA-256 hash function.
Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic example

... | eval n=sha256(field)

sha512(X)

Description

This function computes and returns the secure hash of a string value X based on the FIPS compliant SHA-512 hash function.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic example

... | eval n=sha512(field)

Date and Time functions

The following list contains the functions that you can use to calculate dates and time.

For information about using string and numeric fields in functions, and nesting functions, see Evaluation functions.

In addition to the functions listed in this topic, there are also variables and modifiers that you can use in searches.

- Date and time format variables
- Time modifiers
**now()**

**Description**

This function takes no arguments and returns the time that the search was started.

**Usage**

The `now()` function is often used with other data and time functions.

The time returned by the `now()` function is represented in UNIX time, or in seconds since Epoch time.

When used in a search, this function returns the UNIX time when the search is run. If you want to return the UNIX time when each result is returned, use the `time()` function instead.

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example determines the UNIX time value of the start of yesterday, based on the value of `now()`.

```
... | eval n=relative_time(now(), "-1d@d")
```

**Extended example**

If you are looking for events that occurred within the last 30 minutes you need to calculate the event hour, event minute, the current hour, and the current minute. You use the `now()` function to calculate the current hour (curHour) and current minute (curMin). The event timestamp, in the `_time` field, is used to calculate the event hour (eventHour) and event minute (eventMin). For example:

```
... earliest=-30d | eval eventHour=strftime(_time,"%H") | eval eventMin=strftime(_time,"%M") | eval curHour=strftime(now(),"%H") | eval curMin=strftime(now(),"%M") | where (eventHour=curHour and eventMin > curMin - 30) or (curMin < 30 and eventHour=curHour-1 and eventMin>curMin+30) | bucket _time span=1d | chart count by _time
```
relative_time(X,Y)

Description

This function takes a UNIX time, X, as the first argument and a relative time specifier, Y, as the second argument and returns the UNIX time value of Y applied to X.

Usage

You can use this function with the \texttt{eval}, \texttt{fieldformat}, and \texttt{where} commands, and as part of eval expressions.

Basic example

The following example determines the UNIX time value of the start of yesterday, based on the value of \texttt{now()}.

\texttt{... | eval n=relative_time(now(), "-1d@d")}

strftime(X,Y)

Description

This function takes a UNIX time value, X, as the first argument and renders the time as a string using the format specified by Y. The UNIX time must be in seconds. Use the first 10 digits of a UNIX time to use the time in seconds.

Usage

If the time is in milliseconds, microseconds, or nanoseconds you must convert the time into seconds. You can use the \texttt{pow} function to convert the number.

- To convert from milliseconds to seconds, divide the number by 1000 or 10^3.
- To convert from microseconds to seconds, divide the number by 10^6.
- To convert from nanoseconds to seconds, divide the number by 10^9.

The following search uses the \texttt{pow} function to convert from nanoseconds to seconds:

\texttt{| makeresults | eval StartTimestamp="1521467703049000000" | eval starttime=strftime(StartTimestamp/pow(10,9),"%Y-%m-%dT%H:%M:%S.%Q")}
The results appear on the Statistics tab and look like this:

<table>
<thead>
<tr>
<th>StartTimeStamp</th>
<th>_time</th>
<th>starttime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1521467703049000000</td>
<td>2018-08-10 09:04:00</td>
<td>2018-03-19T06:55:03.049</td>
</tr>
</tbody>
</table>

In these results the `_time` value is the date and time when the search was run.

For a list and descriptions of format options, see Common time format variables.

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of `eval` expressions.

**Basic example**

The following example returns the hour and minute from the `_time` field.

```bash
...| eval hour_min=strftime(_time, "%H:%M")
```

If the `_time` field value is `2018-08-10 11:48:23`, the value returned in the `hour_min` field is `11:48`.

**Extended example**

The following example creates a single result using the `makeresults` command.

```bash
| makeresults
```

For example:

<table>
<thead>
<tr>
<th>_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-08-14 14:00:15</td>
</tr>
</tbody>
</table>

The `_time` field is stored in UNIX time, even though it displays in a human readable format. To convert the UNIX time to some other format, you use the `strftime` function with the date and time format variables. The variables must be in quotations marks.

For example, to return the week of the year that an event occurred in, use the `%V` variable.

```bash
| makeresults | eval week=strftime(_time,"%V")
```

The results are show the value `33` for week.
To return the date and time with subseconds and the time designator (the letter T) that precedes the time components of the format, use the \%Y-%m-%dT%H:%M:%S.%Q variables. For example:

| makeresults | eval mytime=strftime(_time,\"%Y-%m-%dT%H:%M:%S.%Q\") |

The results are:

<table>
<thead>
<tr>
<th>_time</th>
<th>mytime</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-08-14 14:00:15</td>
<td>2018-08-14T14:00:15.000</td>
</tr>
</tbody>
</table>

**strptime(X,Y)**

**Description**

This function takes a time represented by a string X and parses the time into a UNIX timestamp. You use date and time variables to specify the format Y that matches string X.

For example, if string X is 2018-08-13 11:22:33, the format Y must be \%Y-%m-%d \%H:%M:%S. The string X date must be January 1, 1971 or later.

The _time field is in UNIX time. In Splunk Web, the _time field appears in a human readable format in the UI but is stored in UNIX time. If you attempt to use the strptime function on the _time field, no action is performed on the values in the field.

**Usage**

With the strptime function, you must specify the time format of the string X so that the function can convert the string time into the correct UNIX time. The following table shows some examples:

<table>
<thead>
<tr>
<th>String time</th>
<th>Matching time format variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon July 23 2018 17:19:01.89</td>
<td>%a %B %d %Y: %H: %M: %S: %N</td>
</tr>
<tr>
<td>Mon 7/23/2018 17:19:01.89</td>
<td>%a %m/%d/%Y %H: %M: %S: %N</td>
</tr>
<tr>
<td>2018/07/23 17:19:01.89</td>
<td>%Y/%m/%d %H: %M: %S: %N</td>
</tr>
<tr>
<td>2018-07-23T17:19:01.89</td>
<td>%Y-%m-%dT%H: %M: %S: %N</td>
</tr>
</tbody>
</table>

For a list and descriptions of format options, see Common time format variables.
You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

If the values in the `timeStr` field are hours and minutes, such as 11:59, the following example returns the time as a timestamp:

```plaintext
... | eval n=strptime(timeStr, "%H:%M")
```

**Extended example**

This example shows the results of using the `strptime` function. The following search does several things:

- The `gentimes` command generates a set of times with 6 hour intervals. This command returns four fields: `starttime`, `starthuman`, `endtime`, and `endhuman`.
- The `fields` command returns only the `starthuman` and `endhuman` fields.
- The `eval` command takes the string time values in the `starthuman` field and returns the UNIX time that corresponds to the string time values.

```plaintext
| gentimes start=8/13/18 increment=6h | fields starthuman endhuman |
| eval startunix=strptime(starthuman,"%a %B %d %H:%M:%S.%N %Y")
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>starthuman</th>
<th>endhuman</th>
<th>startunix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon Aug 13 00:00:00 2018</td>
<td>Mon Aug 13 05:59:59 2018</td>
<td>534143600.000000</td>
</tr>
<tr>
<td>Mon Aug 13 06:00:00 2018</td>
<td>Mon Aug 13 11:59:59 2018</td>
<td>1534165200.000000</td>
</tr>
<tr>
<td>Mon Aug 13 12:00:00 2018</td>
<td>Mon Aug 13 17:59:59 2018</td>
<td>534186800.000000</td>
</tr>
<tr>
<td>Mon Aug 13 18:00:00 2018</td>
<td>Mon Aug 13 23:59:59 2018</td>
<td>1534208400.000000</td>
</tr>
<tr>
<td>Tue Aug 14 00:00:00 2018</td>
<td>Tue Aug 14 05:59:59 2018</td>
<td>1534230000.000000</td>
</tr>
<tr>
<td>Tue Aug 14 06:00:00 2018</td>
<td>Tue Aug 14 11:59:59 2018</td>
<td>1534251600.000000</td>
</tr>
<tr>
<td>Tue Aug 14 12:00:00 2018</td>
<td>Tue Aug 14 17:59:59 2018</td>
<td>1534273200.000000</td>
</tr>
<tr>
<td>Tue Aug 14 18:00:00 2018</td>
<td>Tue Aug 14 23:59:59 2018</td>
<td>1534294800.000000</td>
</tr>
</tbody>
</table>
**time()**

**Description**

This function returns the wall-clock time, in the UNIX time format, with microsecond resolution.

**Usage**

The value of the `time()` function will be different for each event, based on when that event was processed by the `eval` command.

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

This example shows the results of using the `time()` function. The following search does several things:

- The `gentimes` command generates a set of times with 6 hour intervals. This command returns four fields: `starttime`, `starthuman`, `endtime`, and `endhuman`.
- The `fields` command returns only the `starttime` and `starthuman` fields.
- The first `eval` command takes the numbers in the `starttime` field and returns them with microseconds included.
- The second `eval` command creates the `testtime` field and returns the UNIX time at the instant the result was processed by the `eval` command.

```plaintext
| gentimes start=8/13/18 increment=6h | fields starttime starthuman | eval epoch_time=strptime(starttime,"%s") | eval testtime=time() |
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>starttime</th>
<th>starthuman</th>
<th>epoch_time</th>
<th>testtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1534143600</td>
<td>Mon Aug 13 00:00:00 2018</td>
<td>1534143600.000000</td>
<td>1534376565.299298</td>
</tr>
<tr>
<td>1534165200</td>
<td>Mon Aug 13 06:00:00 2018</td>
<td>1534165200.000000</td>
<td>1534376565.299300</td>
</tr>
<tr>
<td>1534186800</td>
<td>Mon Aug 13 12:00:00 2018</td>
<td>1534186800.000000</td>
<td>1534376565.299302</td>
</tr>
</tbody>
</table>

94
Notice the difference in the microseconds between the values in the `epoch_time` and `test_time` fields. You can see that the `test_time` values increase with each result.

### Informational functions

The following list contains the functions that you can use to return information about a value.

For information about using string and numeric fields in functions, and nesting functions, see [Evaluation functions](#).

#### `isbool(X)`

**Description**

This function takes one argument X and evaluates whether X is a Boolean data type. The function returns TRUE if X is Boolean.

**Usage**

Use this function with other functions that return Boolean data types, such as `cidrmatch` and `mvfind`.

This function cannot be used to determine if field values are "true" or "false" because field values are either string or number data types. Instead, use syntax such as `<fieldname>=true OR <fieldname>=false` to determine field values.
You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**isint(X)**

*Description*

This function takes one argument X and returns TRUE if X is an integer.

*Usage*

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

*Basic examples*

The following example uses the `isint` function with the `if` function. A field, "n", is added to each result with a value of "int" or "not int", depending on the result of the `isint` function. If the value of "field" is a number, the `isint` function returns TRUE and the value adds the value "int" to the "n" field.

```plaintext
... | eval n=if(isint(field),"int", "not int")
```

The following example shows how to use the `isint` function with the `where` command.

```plaintext
... | where isint(field)
```

**isnotnull(X)**

*Description*

This function takes one argument X and returns TRUE if X is not NULL.

*Usage*

This function is useful for checking for whether or not a field (X) contains a value.
You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

The following example uses the `isnotnull` function with the `if` function. A field, "n", is added to each result with a value of "yes" or "no", depending on the result of the `isnotnull` function. If the value of "field" is a number, the `isnotnull` function returns TRUE and the value adds the value "yes" to the "n" field.

```plaintext
... | eval n=if(isnotnull(field),"yes","no")
```

The following example shows how to use the `isnotnull` function with the `where` command.

```plaintext
... | where isnotnull(field)
```

**isnull(X)**

**Description**

This function takes one argument X and returns TRUE if X is NULL.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

The following example uses the `isnull` function with the `if` function. A field, "n", is added to each result with a value of "yes" or "no", depending on the result of the `isnull` function. If there is no value for "field" in a result, the `isnull` function returns TRUE and adds the value "yes" to the "n" field.

```plaintext
... | eval n=if(isnull(field),"yes","no")
```

The following example shows how to use the `isnull` function with the `where` command.
... | where isnull(field)

isnum(X)

Description
This function takes one argument X and returns TRUE if X is a number.

Usage
You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic examples
The following example uses the isnum function with the if function. A field, "n", is added to each result with a value of "yes" or "no", depending on the result of the isnum function. If the value of "field" is a number, the isnum function returns TRUE and the value adds the value "yes" to the "n" field.

... | eval n=if(isnum(field),"yes","no")

The following example shows how to use the isnum function with the where command.

... | where isnum(field)

isstr(X)

Description
This function takes one argument X and returns TRUE if X is a string.

Usage
You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.
Basic examples

The following example uses the isstr function with the if function. A field, "n", is added to each result with a value of "yes" or "no", depending on the result of the isstr function. If the value of "field" is a string, the isstr function returns TRUE and the value adds the value "yes" to the "n" field.

... | eval n=if(isstr(field),"yes","no")

The following example shows how to use the isstr function with the where command.

... | where isstr(field)

typeof(X)

Description

This function takes one argument and returns the data type of the argument.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic examples

The following example takes one argument and returns a string representation of its type. This example returns "NumberStringBoolInvalid"

... | eval n=typeof(12) + typeof("string") + typeof(1==2) + typeof(badfield)

The following example creates a single result using the makeresults command.

| makeresults

For example:
To determine the data type of the `_time` field, use the `eval` command with the `typeof` function. For example:

```
| makeresults | eval t=typeof(_time)
```

The results are:

<table>
<thead>
<tr>
<th>_time</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-08-14 14:00:15</td>
<td>Number</td>
</tr>
</tbody>
</table>

### Mathematical functions

The following list contains the functions that you can use to perform mathematical calculations.

For information about using string and numeric fields in functions, and nesting functions, see Evaluation functions.

**abs(X)**

**Description**

This function takes a number X and returns its absolute value.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example creates a field called `absnum`, whose values are the absolute values of the numeric field `number`.

```
... | eval absnum=abs(number)
```
ceiling(X) or ceil(X)

Description
This function rounds a number X up to the next highest integer.

Usage
You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

You can use the abbreviation ceil(X) instead of the full name of the function.

Basic example
The following example returns n=2.

... | eval n=ceil(1.9)

exact(X)

Description
This function renders the result of a numeric eval calculation with a larger amount of precision in the formatted output.

Usage
You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic example

... | eval n=exact(3.14 * num)

exp(X)

Description
This function takes a number X and returns the exponential function \( e^x \).
**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example returns $y=e^3$.

```
... | eval y=exp(3)
```

**floor(X)**

**Description**

This function rounds a number $X$ down to the nearest whole integer.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example returns 1.

```
... | eval n=floor(1.9)
```

**ln(X)**

**Description**

This function takes a number $X$ and returns its natural logarithm.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.
Basic example

The following example returns the natural logarithm of the values of bytes.

```bash
... | eval lnBytes=ln(bytes)
```

### log(X,Y)

**Description**

This function takes either one or two numeric arguments and returns the logarithm of the first argument X using the second argument Y as the base. If the second argument Y is omitted, this function evaluates the logarithm of number X with base 10.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

```bash
... | eval num=log(number,2)
```

### pi()

**Description**

This function takes no arguments and returns the constant \( \pi \) to 11 digits of precision.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.
**Basic example**

The following example calculates the area of a circle, which is \(\pi()\) multiplied by the radius to the power of 2.

```... | eval area_circle=\pi()*\text{pow}(\text{radius},2)```

**\(\text{pow}(X,Y)\)**

**Description**

This function takes two numeric arguments \(X\) and \(Y\) and returns \(X^Y\), \(X\) to the power of \(Y\).

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example calculates the area of a circle, which is \(\pi()\) multiplied by the radius to the power of 2.

```... | eval area_circle=\pi()*\text{pow}(\text{radius},2)```

**\(\text{round}(X,Y)\)**

**Description**

This function takes one or two numeric arguments \(X\) and \(Y\), returning \(X\) rounded to the amount of decimal places specified by \(Y\). The default is to round to an integer.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.
**Basic examples**

The following example returns n=4.

```
... | eval n=round(3.5)
```

The following example returns n=2.56.

```
... | eval n=round(2.555, 2)
```

**sigfig(X)**

**Description**

This function takes one argument X, a number, and rounds that number to the appropriate number of significant figures.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

The computation for `sigfig` is based on the type of calculation that generates the number.

- For multiplication and division, the result should have the minimum number of significant figures of all of the operands.
- For addition and subtraction, the result should have the same number of decimal places as the least precise number of all of the operands.

For example, the numbers 123.0 and 4.567 contain different precision with the decimal places. The first number is less precise because it has 1 decimal place. The second number is more precise because it has 3 decimal places.

If the calculation is 123.0 + 4.567 = 127.567, then the `sigfig` function returns the fewest number of decimal places. In this example only one decimal place is returned. Because the numbers to the right of the last significant figure are greater than 5, the result returned is 127.6
**Basic example**

The calculation $1.00 \times 1111$ returns the value $1111$, but the following search using the `sigfig` function returns $n=1110$.

```bash
... | eval n=sigfig(1.00*1111)
```

In this example, $1.00$ has 3 significant figures and $1111$ has 4 significant figures. In this example the minimum number of significant figures for all operands is 3. The final result is rounded to 3 digits, returning $n=1110$ and not $1111$.

**sqrt(X)**

**Description**

This function takes one numeric argument $X$ and returns its square root.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example returns 3:

```bash
... | eval n=sqrt(9)
```

**Multivalue eval functions**

The following list contains the functions that you can use on multivalue fields or to return multivalue fields.

For information about using string and numeric fields in functions, and nesting functions, see Evaluation functions.

**commands(X)**
**Description**

This function takes a search string, or field that contains a search string, X and returns a multivalued field containing a list of the commands used in X.

**Usage**

This function is generally not recommended for use except for analysis of audit.log events.

You can use this function with the **eval**, **fieldformat**, and **where** commands, and as part of eval expressions.

**Basic example**

The following example returns a multivalued field X, that contains 'search', 'stats', and 'sort'.

```
... | eval x=commands("search foo | stats count | sort count")
```

**mvappend(X,...)**

**Description**

This function takes an arbitrary number of arguments and returns a multivalue result of all the values. The arguments can be strings, multivalue fields or single value fields.

**Usage**

You can use this function with the **eval**, **fieldformat**, and **where** commands, and as part of eval expressions.

**Basic example**

```
... | eval fullName=mvappend(initial_values, "middle value", last_values)
```
mvcount(MVFIELD)

Description

This function takes a field and returns a count of the values in that field for each result. If the field is a multivalue field, returns the number of values in that field. If the field contains a single value, this function returns 1. If the field has no values, this function returns NULL.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic example

... | eval n=mvcount(multifield)

Extended example

In the following example, the mvcount() function returns the number of email addresses in the To, From, and Cc fields and saves the addresses in the specified "_count" fields.

eventtype="sendmail" | eval To_count=mvcount(split(To,"@"))-1 | eval From_count=mvcount(From) | eval Cc_count= mvcount(split(Cc,"@"))-1

This search takes the values in the To field and uses the split function to separate the email address on the @ symbol. The split function is also used on the Cc field for the same purpose.

If only a single email address exists in the From field, as you would expect, mvcount(From) returns 1. If there is no Cc address, the Cc field might not exist for the event. In that situation mvcount(cc) returns NULL.

mvdedup(X)

Description

This function takes a multivalue field X and returns a multivalue field with its duplicate values removed.
**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

```plaintext
... | eval s=mvedup(mvfield)
```

**mvfilter(X)**

**Description**

This function filters a multivalue field based on an arbitrary Boolean expression X. The Boolean expression X can reference ONLY ONE field at a time.

**Usage**

This function will return NULL values of the field x as well. If you do not want the NULL values, use one of the following expressions:

- `mvfilter(!isnull(x))`
- `mvfilter(isnotnull(x))`

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

The following example returns all of the values in field `email` that end in `.net` or `.org`.

```plaintext
... | eval n=mvfilter(match(email, "\.[.]net\$") OR match(email, "\.[.]org\$"))
```

**mvfind(MVFIELD,"REGEX")**

**Description**

This function tries to find a value in the multivalue field MVFIELD that matches the regular expression in "REGEX". If a match exists, the index of the first matching value is returned (beginning with zero). If no values match, NULL is
returned.

Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

Basic example

```plaintext
... | eval n=mvfind(mymvfield, "err\d")
```

mvindex(MVFIELD,STARTINDEX, ENDINDEX)

Description

This function takes two or three arguments and returns a subset of the multivalue field using the index values provided. The field MVFIELD and the number STARTINDEX are required. The number ENDINDEX is inclusive and optional.

Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

Indexes start at zero. If you have 5 values in the multivalue field, the first value has an index of 0. The second values has an index of 1.

Both the STARTINDEX and ENDINDEX arguments can be negative, where -1 is the last element.

If ENDINDEX is not specified, the function returns only the value at STARTINDEX.

If the indexes are out of range or invalid, the result is NULL.

Basic examples

Because indexes start at zero, the following example returns the third value in "multifield", if the value exists.

```plaintext
... | eval n=mvindex(multifield, 2)
```
**Extended example**

The following search displays at most the last 10 values in the `<field>`.

The STARTINDEX is a range, that starts with the last value, \(-1\). The range is the last 10 values, \(-1-10\). The ENDINDEX is \(-1\), which returns the last value in the field.

- If the multivalue field has 20 values, only the last 10 values are returned.
- If the multivalue field has 3 values, only 3 values are returned.

```... | eval keep=mvindex(<field>,-1-10,-1)```

**mvjoin(MVFIELD,STR)**

**Description**

This function takes two arguments, a multivalue field (MVFIELD) and a string delimiter (STR). The function concatenates the individual values within MVFIELD using the value of STR as a separator.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

You have a multivalue field called "base" that contains the values "1" "2" "3" "4" "5". The values are separated by a space. You want to create a single value field instead, with OR as the delimiter. For example "1 OR 2 OR 3 OR 4 OR 5".

The following search creates the `base` field with the values. The search then creates the `joined` field by using the result of the `mvjoin` function.

```... | eval base=mvrage(1,6), joined=mvjoin('base',' OR ')```

The following example joins together the individual values of "foo" using a semicolon as the delimiter:

```... | eval n=mvjoin(foo, ";")```
mvrange(X,Y,Z)

Description

This function creates a multivalue field for a range of numbers. This function can contain up to three arguments: a starting number X, an ending number Y (which is excluded from the field), and an optional step increment Z. If the increment is a timespan such as 7d, the starting and ending numbers are treated as UNIX time.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic examples

The following example returns a multivalue field with the values 1, 3, 5, 7, 9.

... | eval mv=mvrange(1,11,2)

The following example takes the UNIX timestamp for 1/1/2018 as the start date and the UNIX timestamp for 4/19/2018 as an end date and uses the increment of 7 days.

| makeresults | eval mv=mvrange(1514834731,1524134919,"7d")

This example returns a multivalue field with the UNIX timestamps. The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>mv</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-10 12:31:03</td>
<td>1514834731</td>
</tr>
<tr>
<td></td>
<td>1515439531</td>
</tr>
<tr>
<td></td>
<td>1516044331</td>
</tr>
<tr>
<td></td>
<td>1516649131</td>
</tr>
<tr>
<td></td>
<td>1517253931</td>
</tr>
<tr>
<td></td>
<td>1517858731</td>
</tr>
<tr>
<td></td>
<td>1518463531</td>
</tr>
<tr>
<td></td>
<td>1519068331</td>
</tr>
<tr>
<td></td>
<td>1519673131</td>
</tr>
</tbody>
</table>
mvsort(X)

Description

This function uses a multivalue field X and returns a multivalue field with the values sorted lexicographically.

Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.

- Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically as 10, 100, 70, 9.
- Uppercase letters are sorted before lowercase letters.
- Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

Basic example

```
... | eval s=mvsort(mvfield)
```

mvzip(X,Y,"Z")
**Description**

This function takes two multivalue fields, X and Y, and combines them by stitching together the first value of X with the first value of field Y, then the second with the second, and so on. The third argument, Z, is optional and is used to specify a delimiting character to join the two values. The default delimiter is a comma.

**Usage**

This is similar to the Python `zip` command.

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

```
... | eval nserver=mvzip(hosts,ports)
```

**Extended example**

You can nest several `mvzip` functions together to create a single multivalued field `three_fields` from three separate fields. The pipe (`|`) character is used as the separator between the field values.

```
...| eval three_fields=mvzip(mvzip(field1,field2,"|"),field3,"|")
```

(Thanks to Splunk user cmerriman for this example.)

**split(X,"Y")**

**Description**

This function takes two arguments, field X and delimiting character Y. It splits the values of X on the delimiter Y and returns X as a multivalue field.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

The Splunk software includes a set of multivalue functions. See Multivalue eval functions and Multivalue stats and chart functions.
Basic example

... | eval n=split(foo, ";")

See also

See the following multivalue commands:

makemv, mvcombine, mvexpand, nomv

Statistical eval functions

The following list contains the evaluation functions that you can use to calculate statistics.

For information about using string and numeric fields in functions, and nesting functions, see Evaluation functions.

In addition to these functions, there is a comprehensive set of statistical functions that you can use with the stats, chart, and related commands.

\texttt{max(X,...)}

\textbf{Description}

This function takes an arbitrary number of numeric or string arguments, and returns the maximum. Strings are greater than numbers.

\textbf{Usage}

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

\textbf{Basic example}

The following example returns either "foo" or field, depending on the value of field.

... | eval n=max(1, 3, 6, 7, "foo", field)
**min(X,...)**

*Description*

This function takes an arbitrary number of numeric or string arguments, and returns the minimum. Strings are greater than numbers.

*Usage*

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

*BASIC example*

The following example returns either 1 or field, depending on the value of field.

```plaintext
... | eval n=min(1, 3, 6, 7, "foo", field)
```

**random()**

*Description*

This function takes no arguments and returns a pseudo-random integer ranging from zero to $2^{31}-1$.

*Usage*

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

*BASIC examples*

The following example returns a random integer, such as $0...2147483647$.

```plaintext
... | eval n=random()
```

The following example returns a random number within a specified range. In this example, the random number is between 1 and 100,000.
This example takes a random number and uses the modulo mathematical operator ( % ) to divide the random number by 100000. This ensures that the random number returned is not greater than 100000. The number remaining after the division is increased by 1 to ensure that the number is at least greater than or equal to 1.

**Text functions**

The following list contains the functions that you can use with string values.

For information about using string and numeric fields in functions, and nesting functions, see Evaluation functions.

**len(X)**

*Description*

This function returns the character length of a string X.

*Usage*

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

*Basic example*

```plaintext
... | eval n=len(field)
```

**lower(X)**

*Description*

This function takes one string argument and returns the string in lowercase.
Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

Basic example

The following example returns the value provided by the field `username` in lowercase.

```bash
... | eval username=lower(username)
```

`ltrim(X,Y)`

Description

This function takes one or two arguments X and Y, and returns X with the characters in Y trimmed from the left side. If Y is not specified, spaces and tabs are removed.

Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

Basic example

The following example trims the leading spaces and all of the occurrences of the letter Z from the left side of the string. The value that is returned is `x="abcZZ "`.

```bash
... | eval x=ltrim(" ZZZZabcZZ ", " Z")
```

`replace(X,Y,Z)`

Description

This function returns a string formed by substituting string Z for every occurrence of regex string Y in string X. The third argument Z can also reference groups that are matched in the regex.
Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

To replace a backslash (\) character, you must escape the backslash twice. This is because the `replace` function occurs inside an eval expression. The eval expression perform one level of escaping before passing the regular expression to PCRE. Then PCRE performs its own escaping. See SPL and regular expressions.

Basic example

The following example returns date, with the month and day numbers switched. If the input is 1/14/2017 the return value would be 14/1/2017.

```plaintext
... | eval n=replace(date, "^\d{1,2}/\d{1,2}/", "\2/\1/")
```

`rtrim(X,Y)`

Description

This function takes one or two arguments X and Y, and returns X with the characters in Y trimmed from the right side. If Y is not specified, spaces and tabs are removed.

Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

Basic example

The following example returns `n="ZZZabc"`.

```plaintext
... | eval n=rtrim(" ZZZabcZZ ", " Z")
```
spath(X,Y)

Description

This function takes two arguments, an input source field X and an spath expression Y, that is the XML or JSON formatted location path to the value that you want to extract from X.

Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

If Y is a literal string, it needs quotes, `spath(X,"Y")`. If Y is a field name (with values that are the location paths), it doesn't need quotes. This might result in a multivalued field. Read more about the `spath` command.

Basic example

The following example returns the values of locDesc elements.

```
... | eval locDesc=spath(_raw, "vendorProductSet.product.desc.locDesc")
```

The following example returns the hashtags from a twitter event.

```
index=twitter | eval output=spath(_raw, "entities.hashtags")
```

substr(X,Y,Z)

Description

This function takes either two or three arguments. The required arguments are X, a string, and Y, a numeric. Z is optional and a numeric. This function returns a substring of X, starting at the index specified by Y with the number of characters specified by Z. If Z is not provided, the function returns the rest of the string.

Usage

The indexes follow SQLite semantics; they start at 1. Negative indexes can be used to indicate a start from the end of the string.
You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example concatenates "str" and "ing" together, returning "string":

```plaintext
... | eval n=substr("string", 1, 3) + substr("string", -3)
```

**trim(X,Y)**

**Description**

This function takes one or two arguments X and Y and returns X with the characters in Y trimmed from both sides. If Y is not specified, spaces and tabs are removed.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example returns "abc".

```plaintext
... | eval n=trim(" ZZZZabcZZ ", " Z")
```

**upper(X)**

**Description**

This function takes one string argument and returns the string in uppercase.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.
**Basic example**

The following example returns the value provided by the field `username` in uppercase.

```plaintext
... | eval n=upper(username)
```

**urldecode(X)**

**Description**

This function takes one URL string argument X and returns the unescaped or decoded URL string.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example returns "http://www.splunk.com/download?r=header".

```plaintext
... | eval n=urldecode("http%3A%2F%2Fwww.splunk.com%2Fdownload%3Fr%3Dheader")
```

**Trig and Hyperbolic functions**

The following list contains the functions that you can use to calculate trigonometry and hyperbolic values.

For information about using string and numeric fields in functions, and nesting functions, see Evaluation functions.

**acos(X)**

**Description**

This function computes the arc cosine of X, in the interval [0,pi] radians.
Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

Basic examples

The following example calculates the arc cosine of 0.

```... | eval n=acos(0)```

The following example calculates 180 divided by $\pi$ and multiplies the result by the arc cosine of 0.

```... | eval degrees=acos(0)*180/\pi()```

**acosh(X)**

Description

This function computes the arc hyperbolic cosine of $X$, in radians.

Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

Basic example

```... | eval n=acosh(2)```

**asin(X)**

Description

This function computes the arc sine of $X$, in the interval $[-\pi/2, +\pi/2]$ radians.
**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example calculates the arc sine of 1.

```
... | eval n=asin(1)
```

The following example calculates 180 divided by `pi` and multiplies that by the arc sine of 1.

```
... | eval degrees=asin(1)*180/pi()
```

**asinh(X)**

**Description**

This function computes the arc hyperbolic sine of `X`, in radians.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

```
... | eval n=asinh(1)
```

**atan(X)**

**Description**

This function computes the arc tangent of `X`, in the interval [-pi/2,+pi/2] radians.
**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

```
... | eval n=atan(0.50)
```

**atan2(Y, X)**

**Description**

This function computes the arc tangent of Y, X in the interval [-π,+π] radians.

Y is a value that represents the proportion of the y-coordinate. X is the value that represents the proportion of the x-coordinate.

To compute the value, the function takes into account the sign of both arguments to determine the quadrant.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

```
... | eval n=atan2(0.50, 0.75)
```

**atanh(X)**

**Description**

This function computes the arc hyperbolic tangent of X, in radians.
**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

```plaintext
... | eval n=atanh(0.500)
```

**cos(X)**

**Description**

This function computes the cosine of an angle of X radians.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic example**

The following example calculates the cosine of -1.

```plaintext
... | eval n=cos(-1)
```

The following example calculates the cosine of `pi`.

```plaintext
... | eval n=cos(pi())
```

**cosh(X)**

**Description**

This function computes the hyperbolic cosine of X radians.
Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

Basic example

```plaintext
... | eval n=cosh(1)
```

hypot(X,Y)

Description

This function computes the hypotenuse of a right-angled triangle whose legs are X and Y.

The function returns the square root of the sum of the squares of X and Y, as described in the Pythagorean theorem.

Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

Basic example

```plaintext
... | eval n=hypot(3,4)
```

sin(X)

Description

This function computes the sine of X.

Usage

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.
**Basic examples**

The following example calculates the sine of 1.

```
... | eval n=sin(1)
```

The following search calculates the sine of pi divided by 180 and then multiplied by 90.

```
... | eval n=sin(90 * pi()/180)
```

**sinh(X)**

*Description*

This function computes the hyperbolic sine of X.

*Usage*

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

*Basic example*

```
... | eval n=sinh(1)
```

**tan(X)**

*Description*

This function computes the tangent of X.

*Usage*

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.
Basic example

... | eval n=tan(1)

tanh(X)

Description

This function computes the hyperbolic tangent of X.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic example

... | eval n=tanh(1)
Statistical and Charting Functions

Statistical and charting functions

You can use the statistical and charting functions with the `chart`, `stats`, and `timechart` commands.

Support for related commands

The functions can also be used with related statistical and charting commands. The following table lists the commands supported by the statistical and charting functions and the related command that can also use these functions.

<table>
<thead>
<tr>
<th>Command</th>
<th>Supported related commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>chart</td>
<td>• <code>sichart</code></td>
</tr>
<tr>
<td></td>
<td>• <code>eventstats</code></td>
</tr>
<tr>
<td></td>
<td>• <code>streamstats</code></td>
</tr>
<tr>
<td></td>
<td>• <code>geostats</code></td>
</tr>
<tr>
<td></td>
<td>• <code>sistats</code></td>
</tr>
<tr>
<td></td>
<td>• For the <code>tstats</code> and the <code>mstats</code> commands, see the documentation for each command for a list of the supported functions.</td>
</tr>
<tr>
<td>timechart</td>
<td>• <code>sitimechart</code></td>
</tr>
</tbody>
</table>

Functions that you can use to create sparkline charts are noted in the documentation for each function. Sparkline is a function that applies to only the `chart` and `stats` commands, and allows you to call other functions. For more information, see Add sparklines to search results in the Search Manual.

How field values are processed

Most of the statistical and charting functions expect the field values to be numbers. All of the values are processed as numbers, and any non-numeric values are ignored.

The following functions process the field values as literal string values, even though the values are numbers.
For example, you use the `distinct_count` function and the field contains values such as "1", "1.0", and "01". Each value is considered a distinct string value.

The only exceptions are the `max` and `min` functions. These functions process values as numbers if possible. For example, the values "1", "1.0", and "01" are processed as the same numeric value.

**Supported functions and syntax**

The following table is a quick reference of the supported statistical and charting functions. This table lists the syntax and provides a brief description for each of the functions. Use the links in the table to learn more about each function examples, and to see examples.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate functions</td>
<td><code>avg(X)</code></td>
<td>Returns the average of the values in the field X.</td>
</tr>
<tr>
<td></td>
<td><code>count(X)</code></td>
<td>Returns the number of occurrences where the field that you specify contains any value (is not empty). You can also count the occurrences of a specific value in the field by using the <code>eval</code> command with the <code>count</code> function. For example: <code>count eval(field_name=&quot;value&quot;)</code>.</td>
</tr>
<tr>
<td></td>
<td><code>distinct_count(X)</code></td>
<td>Returns the count of distinct values in the field X.</td>
</tr>
<tr>
<td></td>
<td><code>estdc(X)</code></td>
<td>Returns the estimated count of the distinct values in the field X.</td>
</tr>
<tr>
<td></td>
<td><code>estdc_error(X)</code></td>
<td>Returns the theoretical error of the estimated count of the distinct values in the field X. The error represents a ratio of the <code>absolute_value(estimate_distinct_count - real_distinct_count)/real_distinct_count</code>.</td>
</tr>
<tr>
<td></td>
<td><code>max(X)</code></td>
<td>Returns the maximum value of the field X. If the values of X are non-numeric, the maximum value is found using lexicographical ordering.</td>
</tr>
<tr>
<td>Supported functions and syntax</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>mean(X)</td>
<td>Returns the arithmetic mean of the field X.</td>
<td></td>
</tr>
<tr>
<td>median(X)</td>
<td>Returns the middle-most value of the field X.</td>
<td></td>
</tr>
<tr>
<td>min(X)</td>
<td>Returns the minimum value of the field X. If the values of X are non-numeric, the minimum value is found using lexicographical ordering.</td>
<td></td>
</tr>
<tr>
<td>mode(X)</td>
<td>Returns the most frequent value of the field X.</td>
<td></td>
</tr>
<tr>
<td>percentile&lt;X&gt;(Y)</td>
<td>Returns the X-th percentile value of the numeric field Y. Valid values of X are integers from 1 to 99. Additional percentile functions are upperperc&lt;X&gt;(Y) and exactperc&lt;X&gt;(Y).</td>
<td></td>
</tr>
<tr>
<td>range(X)</td>
<td>Returns the difference between the maximum and minimum values of the field X ONLY IF the values of X are numeric.</td>
<td></td>
</tr>
<tr>
<td>stdev(X)</td>
<td>Returns the sample standard deviation of the field X.</td>
<td></td>
</tr>
<tr>
<td>stdevp(X)</td>
<td>Returns the population standard deviation of the field X.</td>
<td></td>
</tr>
<tr>
<td>sum(X)</td>
<td>Returns the sum of the values of the field X.</td>
<td></td>
</tr>
<tr>
<td>sumsq(X)</td>
<td>Returns the sum of the squares of the values of the field X.</td>
<td></td>
</tr>
<tr>
<td>var(X)</td>
<td>Returns the sample variance of the field X.</td>
<td></td>
</tr>
<tr>
<td>varp(X)</td>
<td>Returns the population variance of the field X.</td>
<td></td>
</tr>
<tr>
<td>first(X)</td>
<td>Returns the first seen value of the field X. In general, the first seen value of the field is the most recent instance of this field, relative to the input order of events into the stats command.</td>
<td></td>
</tr>
<tr>
<td>last(X)</td>
<td>Returns the last seen value of the field X. In</td>
<td></td>
</tr>
<tr>
<td>Type of function</td>
<td>Supported functions and syntax</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Multivalue stats and chart functions</strong></td>
<td></td>
<td>general, the last seen value of the field is the oldest instance of this field relative to the input order of events into the stats command.</td>
</tr>
<tr>
<td>list(X)</td>
<td>Returns a list of up to 100 values of the field X as a multivalue entry. The order of the values reflects the order of input events.</td>
<td></td>
</tr>
<tr>
<td>values(X)</td>
<td>Returns the list of all distinct values of the field X as a multivalue entry. The order of the values is lexicographical.</td>
<td></td>
</tr>
<tr>
<td>earliest(X)</td>
<td>Returns the chronologically earliest (oldest) seen occurrence of a value of a field X.</td>
<td></td>
</tr>
<tr>
<td>earliest_time(X)</td>
<td>Returns the UNIX time of the earliest (oldest) occurrence of a value of the field. Used in conjunction with earliest(x), latest(x), and latest_time(x) to calculate the rate of increase for an accumulating counter.</td>
<td></td>
</tr>
<tr>
<td>latest(X)</td>
<td>Returns the chronologically latest (most recent) seen occurrence of a value of a field X.</td>
<td></td>
</tr>
<tr>
<td>latest_time(X)</td>
<td>Returns the UNIX time of the latest (most recent) occurrence of a value of the field. Used in conjunction with earliest(x), earliest_time(x), and latest(x) to calculate the rate of increase for an accumulating counter.</td>
<td></td>
</tr>
<tr>
<td>per_day(X)</td>
<td>Returns the values of field X, or eval expression X, for each day.</td>
<td></td>
</tr>
<tr>
<td>per_hour(X)</td>
<td>Returns the values of field X, or eval expression X, for each hour.</td>
<td></td>
</tr>
<tr>
<td>per_minute(X)</td>
<td>Returns the values of field X, or eval expression X, for each minute.</td>
<td></td>
</tr>
<tr>
<td>per_second(X)</td>
<td>Returns the values of field X, or eval expression X, for each second.</td>
<td></td>
</tr>
<tr>
<td>rate(X)</td>
<td>Returns the per-second rate change of the value of the field. Represents ((\text{latest}(X) - \text{earliest}(X)) / (\text{latest}<em>\text{time}(X) - \text{earliest}</em>\text{time}(X))).</td>
<td></td>
</tr>
</tbody>
</table>
earliest_time(X)) Requires the earliest(X) and latest(X) values of the field to be numerical, and the earliest_time(X) and latest_time(X) values to be different.

See also

Evaluation functions

stats, chart, timechart, eventstats, streamstats, geostats

Answers

Have questions? Visit Splunk Answers and search for a specific function or command.

Aggregate functions

Aggregate functions summarize the values from each event to create a single, meaningful value. Common aggregate functions include Average, Count, Minimum, Maximum, Standard Deviation, Sum, and Variance.

Most aggregate functions are used with numeric fields. However, there are some functions that you can use with either alphabetic string fields or numeric fields. The function descriptions indicate which functions you can use with alphabetic strings.

For an overview, see statistical and charting functions.

avg(X)

Description

Returns the average of the values of field X.
Usage

You can use this function with the chart, stats, and timechart commands, and also with sparkline() charts.

For a list of the related statistical and charting commands that you can use with this function, see Statistical and charting functions.

Basic examples

The following example returns the average (mean) "size" for each distinct "host".

... | stats avg(size) BY host

The following example returns the average "thruput" of each "host" for each 5 minute time span.

... | bin _time span=5m | stats avg(thruput) BY _time host

The following example charts the ratio of the average (mean) "size" to the maximum "delay" for each distinct "host" and "user" pair.

... | chart eval(avg(size)/max(delay)) AS ratio BY host user

The following example displays a timechart of the average of cpu_seconds by processor, rounded to 2 decimal points.

... | timechart eval(round(avg(cpu_seconds),2)) BY processor

Extended example

Chart the average number of events in a transaction, based on transaction duration.

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.
1. Run the following search to create a chart to show the average number of events in a transaction based on the duration of the transaction.

```
sourcetype=access_* status=200 action=purchase | transaction clientip maxspan=30m | chart avg(eventcount) by duration span=log2
```

The `transaction` command adds two fields to the results: `duration` and `eventcount`. The `eventcount` field tracks the number of events in a single transaction.

In this search, the transactions are piped into the `chart` command. The `avg()` function is used to calculate the average number of events for each duration. Because the duration is in seconds and you expect there to be many values, the search uses the `span` argument to bucket the duration into bins using logarithm with a base of 2.

2. Use the field format option to enable number formatting.

3. Click the **Visualization** tab and change the display to a pie chart.
Each wedge of the pie chart represents a duration for the event transactions. You can hover over a wedge to see the average values.

**count(X) or c(X)**

**Description**

Returns the number of occurrences of the field X. To indicate a specific field value to match, format X as eval(field="value"). Processes field values as strings. To use this function, you can specify count(X), or the abbreviation c(X).

**Usage**

You can use the count(X) function with the chart, stats, and timechart commands, and also with sparkline() charts.

**Basic examples**

The following example returns the count of events where the status field has the value "404". This example uses an eval expression with the count function.

```plaintext
...count(eval(status="404")) AS count_status BY sourcetype
```

The following example separates search results into 10 bins and returns the count of raw events for each bin.

```plaintext
... | bin size bins=10 | stats count(_raw) BY size
```

The following example generates a sparkline chart to count the events that use the _raw field.

```plaintext
... sparkline(count)
```

The following example generates a sparkline chart to count the events that have the user field.

```plaintext
... sparkline(count(user))
```
The following example uses the `timechart` command to count the events where the `action` field contains the value `purchase`.

```
sourcetype=access_* | timechart count(eval(action="purchase")) BY productName usenull=f useother=f
```

**Extended examples**

**Count the number of earthquakes that occurred for each magnitude range**

This search uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the USGS Earthquake Feeds and upload the file to your Splunk instance. This example uses the All Earthquakes data from the past 30 days.

1. Run the following search to calculate the number of earthquakes that occurred in each magnitude range. This data set is comprised of events over a 30-day period.

```
source=all_month.csv | chart count AS "Number of Earthquakes" BY mag span=1 | rename mag AS "Magnitude Range"
```

- This search uses `span=1` to define each of the ranges for the magnitude field, `mag`.
- The `rename command` is then used to rename the field to "Magnitude Range".

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>Magnitude Range</th>
<th>Number of Earthquakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1-0</td>
<td>18</td>
</tr>
<tr>
<td>0-1</td>
<td>2088</td>
</tr>
<tr>
<td>1-2</td>
<td>3005</td>
</tr>
<tr>
<td>2-3</td>
<td>1026</td>
</tr>
<tr>
<td>3-4</td>
<td>194</td>
</tr>
<tr>
<td>4-5</td>
<td>452</td>
</tr>
</tbody>
</table>
### Magnitude Range

<table>
<thead>
<tr>
<th>Magnitude Range</th>
<th>Number of Earthquakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-4</td>
<td>109</td>
</tr>
<tr>
<td>6-7</td>
<td>11</td>
</tr>
<tr>
<td>7-8</td>
<td>3</td>
</tr>
</tbody>
</table>

**Count the number of different page requests for each Web server**

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to **get the tutorial data into Splunk**. Use the time range **All time** when you run the search.

1. Run the following search to use the `chart` command to determine the number of different page requests, GET and POST, that occurred for each Web server.

   ```splunk
   sourcetype=access_* | chart count(eval(method="GET")) AS GET, count(eval(method="POST")) AS POST BY host
   ```

   This example uses `eval` expressions to specify the different field values for the `stats` command to count. The first clause uses the `count()` function to count the Web access events that contain the `method` field value `GET`. Then, using the `AS` keyword, the field that represents these results is renamed `GET`.

   The second clause does the same for POST events. The counts of both types of events are then separated by the web server, using the `BY` clause with the `host` field.

   The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>host</th>
<th>GET</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>www1</td>
<td>8431</td>
<td>5197</td>
</tr>
<tr>
<td>www2</td>
<td>8097</td>
<td>4815</td>
</tr>
<tr>
<td>www3</td>
<td>8338</td>
<td>4654</td>
</tr>
</tbody>
</table>

2. Click the **Visualization** tab. If necessary, format the results as a column chart. This chart displays the total count of events for each event type, GET or POST, based on the `host` value.
distinct_count(X) or dc(X)

**Description**

Returns the count of distinct values of the field X. This function processes field values as strings. To use this function, you can specify `distinct_count(X)`, or the abbreviation `dc(X)`.

**Usage**

You can use this function with the `chart`, `stats`, and `timechart` commands, and also with `sparkline()` charts.

**Basic examples**

The following example removes duplicate results with the same "host" value and return the total count of the remaining results.

```plaintext
... | stats dc(host)
```

The following example generates sparklines for the distinct count of devices and renames the field, "numdevices".

```plaintext
...sparkline(dc(device)) AS numdevices
```

The following example counts the distinct sources for each sourcetype, and buckets the count for each five minute spans.

```plaintext
...sparkline(dc(source),5m) BY sourcetype
```
Extended example

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

1. Run the following search to count the number of different customers who purchased something from the Buttercup Games online store yesterday. The search organizes the count by the type of product (accessories, t-shirts, and type of games) that customers purchased.

```
sourcetype=access_* action=purchase | stats dc(clientip) BY categoryId
```

♦ This example first searches for purchase events, action=purchase.
♦ These results are piped into the stats command and the dc() function counts the number of different users who make purchases.
♦ The BY clause is used to break up this number based on the different category of products, the categoryId.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>categoryId</th>
<th>dc(clientip)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESSORIES</td>
<td>37</td>
</tr>
<tr>
<td>ARCADE</td>
<td>58</td>
</tr>
<tr>
<td>NULL</td>
<td>8</td>
</tr>
<tr>
<td>SHOOTER</td>
<td>31</td>
</tr>
<tr>
<td>SIMULATION</td>
<td>34</td>
</tr>
<tr>
<td>SPORTS</td>
<td>13</td>
</tr>
<tr>
<td>STRATEGY</td>
<td>74</td>
</tr>
<tr>
<td>TEE</td>
<td>38</td>
</tr>
</tbody>
</table>

`estdc(X)`
**Description**

Returns the estimated count of the distinct values of the field X. This function processes field values as strings. The string values 1.0 and 1 are considered distinct values and counted separately.

**Usage**

You can use this function with the `chart`, `stats`, and `timechart` commands.

**Basic examples**

The following example removes duplicate results with the same "host" value and returns the estimated total count of the remaining results.

```plaintext
... | stats estdc(host)
```

The following example generates sparklines for the estimated distinct count of devices and renames the field, "numdevices".

```plaintext
...sparkline(estdc(device)) AS numdevices
```

The following example estimates the distinct count for the sources for each sourcetype. The results are displayed for each five minute span in sparkline charts.

```plaintext
...sparkline(estdc(source),5m) BY sourcetype
```

**estdc_error(X)**

**Description**

Returns the theoretical error of the estimated count of the distinct values of the field X. The error represents a ratio of the absolute_value(estimate_distinct_count - real_distinct_count)/real_distinct_count. This function processes field values as strings.
**Usage**

You can use this function with the `chart`, `stats`, and `timechart` commands.

**Basic examples**

The following example determines the error ratio for the estimated distinct count of the "host" values.

```
... | stats estdc_error(host)
```

**exactperc<X>(Y)**

**Description**

Returns a percentile value of the numeric field Y. See the `perc<X>(Y)` function.

**max(X)**

**Description**

Returns the maximum value of the field X. If the values of X are non-numeric, the maximum value is found using lexicographical ordering.

Processes field values as numbers if possible, otherwise processes field values as strings.

**Usage**

You can use this function with the `chart`, `stats`, and `timechart` commands, and also with `sparkline()` charts.

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.

- Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically as 10, 100, 70, 9.
- Uppercase letters are sorted before lowercase letters.
- Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.
Basic examples

This example returns the maximum value of "size".

max(size)

Extended example

Calculate aggregate statistics for the magnitudes of earthquakes in an area

This search uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the USGS Earthquake Feeds and upload the file to your Splunk instance. This example uses the All Earthquakes data from the past 30 days.

1. Search for earthquakes in and around California. Calculate the number of earthquakes that were recorded. Use statistical functions to calculate the minimum, maximum, range (the difference between the min and max), and average magnitudes of the recent earthquakes. List the values by magnitude type.

    source=all_month.csv place=*California* | stats count, max(mag), min(mag), range(mag), avg(mag) BY magType

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>magType</th>
<th>count</th>
<th>max(mag)</th>
<th>min(mag)</th>
<th>range(mag)</th>
<th>avg(mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>123</td>
<td>2.8</td>
<td>0.0</td>
<td>2.8</td>
<td>0.549593</td>
</tr>
<tr>
<td>MbLg</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000000</td>
</tr>
<tr>
<td>Md</td>
<td>1565</td>
<td>3.2</td>
<td>0.1</td>
<td>3.1</td>
<td>1.056486</td>
</tr>
<tr>
<td>Me</td>
<td>2</td>
<td>2.0</td>
<td>1.6</td>
<td>.04</td>
<td>1.800000</td>
</tr>
<tr>
<td>Ml</td>
<td>1202</td>
<td>4.3</td>
<td>-0.4</td>
<td>4.7</td>
<td>1.226622</td>
</tr>
<tr>
<td>Mw</td>
<td>6</td>
<td>4.9</td>
<td>3.0</td>
<td>1.9</td>
<td>3.650000</td>
</tr>
<tr>
<td>ml</td>
<td>10</td>
<td>1.56</td>
<td>0.19</td>
<td>1.37</td>
<td>0.934000</td>
</tr>
</tbody>
</table>
mean(X)

Description

Returns the arithmetic mean of the field X.

The mean values should be exactly the same as the values calculated using the \texttt{\texttt{avg()}} function.

Usage

You can use this function with the \texttt{chart}, \texttt{stats}, and \texttt{timechart} commands, and also with \texttt{sparkline()} charts.

Basic examples

The following example returns the mean of "kbps" values:

\[
\ldots \text{mean(kbps)}
\]

Extended example

This search uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the \texttt{USGS Earthquake Feeds} and upload the file to your Splunk instance. This example uses the \texttt{All Earthquakes} data from the past 30 days.

1. Run the following search to find the mean, standard deviation, and variance of the magnitudes of recent quakes by magnitude type.

\texttt{source=usgs place=*California* | stats count mean(mag), stdev(mag), var(mag) BY magType}

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>magType</th>
<th>count</th>
<th>mean(mag)</th>
<th>std(mag)</th>
<th>var(mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>123</td>
<td>0.549593</td>
<td>0.356985</td>
<td>0.127438</td>
</tr>
<tr>
<td>MbLg</td>
<td>1</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>magType</td>
<td>count</td>
<td>mean(mag)</td>
<td>std(mag)</td>
<td>var(mag)</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>Md</td>
<td>1565</td>
<td>1.056486</td>
<td>0.580042</td>
<td>0.336449</td>
</tr>
<tr>
<td>Me</td>
<td>2</td>
<td>1.800000</td>
<td>0.346410</td>
<td>0.120000</td>
</tr>
<tr>
<td>Ml</td>
<td>1202</td>
<td>1.226622</td>
<td>0.629664</td>
<td>0.396476</td>
</tr>
<tr>
<td>Mw</td>
<td>6</td>
<td>3.650000</td>
<td>0.716240</td>
<td>0.513000</td>
</tr>
<tr>
<td>ml</td>
<td>10</td>
<td>0.934000</td>
<td>0.560401</td>
<td>0.314049</td>
</tr>
</tbody>
</table>

**median(X)**

*Description*

Returns the middle-most value of the field X.

*Usage*

You can use this function with the `chart`, `stats`, and `timechart` commands.

If you have an even number of events, by default the median calculation is approximated to the higher of the two values. To receive a more accurate median value with an even number of events, change the `perc_method` in the `limits.conf` file.

1. See How to edit a configuration file in the *Admin manual*.

   Only users with file system access, such as system administrators, can edit the configuration files. Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

2. In the `[stats | sistats]` stanza, change the `perc_method` setting to `interpolated`.

If you are using Splunk Cloud and want to edit the configuration file, file a Support ticket.

*Basic examples*
\textbf{min(X)}

\textit{Description}

Returns the minimum value of the field X. If the values of X are non-numeric, the minimum value is found using lexicographical ordering.

This function processes field values as numbers if possible, otherwise processes field values as strings.

\textit{Usage}

You can use this function with the \texttt{chart}, \texttt{stats}, and \texttt{timechart} commands.

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.

- Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically as 10, 100, 70, 9.
- Uppercase letters are sorted before lowercase letters.
- Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

\textit{Basic examples}

The following example returns the minimum size and maximum size of the HotBucketRoller component in the _internal index.

\begin{verbatim}
index=_internal component=HotBucketRoller | stats min(size), max(size)
\end{verbatim}

The following example returns a list of processors and calculates the minimum cpu\_seconds and the maximum cpu\_seconds.

\begin{verbatim}
index=_internal | chart min(cpu\_seconds), max(cpu\_seconds) BY processor
\end{verbatim}

\textit{Extended example}

See the \texttt{Extended example for the max() function}. That example includes the \texttt{min()} function.
**mode(X)**

*Description*

Returns the most frequent value of the field X.

Processes field values as strings.

*Usage*

You can use this function with the `chart`, `stats`, and `timechart` commands.

*Basic examples*

**perc<X>(Y)**

*Description*

There are three different percentile functions:

- `perc<X>(Y)` (or the abbreviation `p<X>(Y)`)
- `upperperc<X>(Y)`
- `exactperc<X>(Y)`

Return the X-th percentile value of the numeric field Y. Valid values of X are floating point numbers between 0 and 100, such as 99.95.

The percentile functions process field values as strings.

Use the `perc<X>(Y)` function to calculate an approximate threshold, such that of the values in field Y, X percent fall below the threshold.

The `perc` and `upperperc` functions give approximate values for the integer percentile requested. The approximation algorithm that is used, which is based on dynamic compression of a radix tree, provides a strict bound of the actual value for any percentile.

- The `perc` function returns a single number that represents the lower end of that range.
- The `upperperc` function gives the approximate upper bound.
- The `exactperc` function provides the exact value, but will be very expensive for high cardinality fields. The `exactperc` function could consume a large amount of memory in the search head.
**Usage**

You can use this function with the `chart`, `stats`, and `timechart` commands.

**Basic examples**

Consider this list of values $Y = \{10, 9, 8, 7, 6, 5, 4, 3, 2, 1\}$.

The following example returns 5.5.

```
...| stats perc50(Y)
```

The following example returns 9.55.

```
...| stats perc95(Y)
```

**range(X)**

**Description**

Returns the difference between the max and min values of the field $X$. The values of field $X$ must be numeric.

**Usage**

You can use this function with the `chart`, `stats`, and `timechart` commands, and also with `sparkline()` charts.

**Basic example**

This example uses events that list the numeric sales for each product and quarter, for example:

<table>
<thead>
<tr>
<th>products</th>
<th>quarter</th>
<th>sales</th>
<th>quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
<td>QTR1</td>
<td>1200</td>
<td>1000</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR1</td>
<td>1400</td>
<td>1550</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR1</td>
<td>1650</td>
<td>1275</td>
</tr>
<tr>
<td>ProductA</td>
<td>QTR2</td>
<td>1425</td>
<td>1300</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR2</td>
<td>1175</td>
<td>1425</td>
</tr>
<tr>
<td>products</td>
<td>quarter</td>
<td>sales</td>
<td>quota</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR2</td>
<td>1550</td>
<td>1450</td>
</tr>
<tr>
<td>ProductA</td>
<td>QTR3</td>
<td>1300</td>
<td>1400</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR3</td>
<td>1250</td>
<td>1125</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR3</td>
<td>1375</td>
<td>1475</td>
</tr>
<tr>
<td>ProductA</td>
<td>QTR4</td>
<td>1550</td>
<td>1300</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR4</td>
<td>1700</td>
<td>1225</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR4</td>
<td>1625</td>
<td>1350</td>
</tr>
</tbody>
</table>

It is easiest to understand the range if you also determine the min and max values. To determine the range of sales by product, run this search:

`source="addtotalsData.csv" | chart sum(sales) min(sales) max(sales) range(sales) BY products`

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>quarter</th>
<th>sum(sales)</th>
<th>min(sales)</th>
<th>max(sales)</th>
<th>range(sales)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QTR1</td>
<td>4250</td>
<td>1200</td>
<td>1650</td>
<td>450</td>
</tr>
<tr>
<td>QTR2</td>
<td>4150</td>
<td>1175</td>
<td>1550</td>
<td>375</td>
</tr>
<tr>
<td>QTR3</td>
<td>3925</td>
<td>1250</td>
<td>1375</td>
<td>125</td>
</tr>
<tr>
<td>QTR4</td>
<td>4875</td>
<td>1550</td>
<td>1700</td>
<td>150</td>
</tr>
</tbody>
</table>

The range(sales) is the max(sales) minus the min(sales).

**Extended example**

See the Extended example for the `max()` function. That example includes the `range()` function.

**stdev(X)**

**Description**

Returns the sample standard deviation of the field X.

**Usage**

You can use this function with the `chart`, `stats`, and `timechart` commands, and also with `sparkline()` charts.
**Basic examples**

This example returns the standard deviation of wildcarded fields "*delay" which can apply to both, "delay" and "xdelay".

\[ \text{stdev(*delay)} \]

**Extended example**

This search uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the USGS Earthquake Feeds and upload the file to your Splunk instance. This example uses the All Earthquakes data from the past 30 days.

1. Run the following search to find the mean, standard deviation, and variance of the magnitudes of recent quakes by magnitude type.

   \[ \text{source=usgs place=*California* | stats count mean(mag), stdev(mag), var(mag) BY magType} \]

   The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>magType</th>
<th>count</th>
<th>mean(mag)</th>
<th>std(mag)</th>
<th>var(mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>123</td>
<td>0.549593</td>
<td>0.356985</td>
<td>0.127438</td>
</tr>
<tr>
<td>MbLg</td>
<td>1</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>Md</td>
<td>1565</td>
<td>1.056486</td>
<td>0.580042</td>
<td>0.336449</td>
</tr>
<tr>
<td>Me</td>
<td>2</td>
<td>1.800000</td>
<td>0.346410</td>
<td>0.120000</td>
</tr>
<tr>
<td>Ml</td>
<td>1202</td>
<td>1.226622</td>
<td>0.629664</td>
<td>0.396476</td>
</tr>
<tr>
<td>Mw</td>
<td>6</td>
<td>3.650000</td>
<td>0.716240</td>
<td>0.513000</td>
</tr>
<tr>
<td>ml</td>
<td>10</td>
<td>0.934000</td>
<td>0.560401</td>
<td>0.314049</td>
</tr>
</tbody>
</table>

\[ \text{stdevp(X)} \]
**Description**

Returns the population standard deviation of the field X.

**Usage**

You can use this function with the chart, stats, and timechart commands, and also with sparkline() charts.

**Basic examples**

**Extended example**

`sum(X)`

**Description**

Returns the sum of the values of the field X.

**Usage**

You can use this function with the chart, stats, and timechart commands, and also with sparkline() charts.

**Basic examples**

`sum(eval(date_hour * date_minute))`

**sumsq(X)**

**Description**

Returns the sum of the squares of the values of the field X.

**Usage**

You can use this function with the chart, stats, and timechart commands, and also with sparkline() charts.
Basic examples

upperperc<X>(Y)

Description

Returns a percentile value of the numeric field Y. See the \texttt{perc<X>(Y)} function.

\texttt{var(X)}

Description

Returns the sample variance of the field X.

Usage

You can use this function with the \texttt{chart}, \texttt{stats}, and \texttt{timechart} commands, and also with \texttt{sparkline()} charts.

Basic examples

Extended example

See the Extended example for the \texttt{mean()} function. That example includes the \texttt{var()} function.

\texttt{varp(X)}

Description

Returns the population variance of the field X.

Usage

You can use this function with the \texttt{chart}, \texttt{stats}, and \texttt{timechart} commands, and also with \texttt{sparkline()} charts.

Basic examples
Event order functions

Use the event order functions to return events in chronological or timestamp order.

The following table lists the timestamps from a set of events returned from a search. This table identifies which event is returned when you use the `first` and `last` event order functions, and compares them with the `earliest` and `latest` functions, which you can read more about at Time functions.

<table>
<thead>
<tr>
<th>_time</th>
<th>Event order function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-28</td>
<td>first</td>
<td>This event is the first event in the search results. But this event is not chronologically the earliest event.</td>
</tr>
<tr>
<td>00:15:05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018-05-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00:15:04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018-04-30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00:15:02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018-04-28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00:15:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018-05-01</td>
<td>latest</td>
<td>This event is chronologically the latest event in the search results.</td>
</tr>
<tr>
<td>00:15:05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018-04-27</td>
<td>earliest</td>
<td>This event is both the chronologically earliest event and the last event in the search results.</td>
</tr>
<tr>
<td>00:15:01</td>
<td>last</td>
<td></td>
</tr>
</tbody>
</table>

See Overview of statistical and charting functions.

**first(X)**

*Description*

Returns the first seen value of the field X. In general, the first seen value of the field is the most recent instance of this field, relative to the input order of events into the `stats` command.
Usage

- To locate the first value based on time order, use the earliest function instead.
- Works best when the search includes the sort command immediately before the statistics or charting command.
- This function processes field values as strings.
- You can use the first(X) function with the chart, stats, and timechart commands.

Basic example

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

You run the following search to locate invalid user login attempts against a specific sshd (Secure Shell Daemon). You use the table command to see the values in the _time, source, and _raw fields.

sourcetype=secure invalid user "sshd[5258]" | table _time source _raw

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>source</th>
<th>_raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-28 00:15:05 tutorialdata.zip:/mailsv/secure.log</td>
<td>Mon Apr 28 2018 00:15:05 mailsv1 sshd[5258]: Failed password for invalid user tomcat from 67.170.226.218 port 1490 ssh2</td>
<td></td>
</tr>
<tr>
<td>2018-05-01 00:15:04 tutorialdata.zip:/www2/secure.log</td>
<td>Thu May 01 2018 00:15:04 www2 sshd[5258]: Failed password for invalid user brian from 130.253.37.97 port 4284 ssh2</td>
<td></td>
</tr>
<tr>
<td>2018-04-30 00:15:02 tutorialdata.zip:/www3/secure.log</td>
<td>Wed Apr 30 2018 00:15:02 www3 sshd[5258]: Failed password for invalid user operator from 222.169.224.226 port 1711</td>
<td></td>
</tr>
<tr>
<td>_time</td>
<td>source</td>
<td>_raw</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>2018-04-28 00:15:01</td>
<td>tutorialdata.zip:/www1/secure.log</td>
<td>Mon Apr 28 2018 00:15:01 www1 sshd[5258]: Failed password for invalid user rightscale from 87.194.216.51 port 3361 ssh2</td>
</tr>
<tr>
<td>2018-05-01 00:15:05</td>
<td>tutorialdata.zip:/mailsv/secure.log</td>
<td>Thu May 01 2018 00:15:05 mailsv1 sshd[5258]: Failed password for invalid user testuser from 194.8.74.23 port 3626 ssh2</td>
</tr>
<tr>
<td>2018-04-27 00:15:01</td>
<td>tutorialdata.zip:/www1/secure.log</td>
<td>Sun Apr 27 2018 00:15:01 www1 sshd[5258]: Failed password for invalid user redmine from 91.208.184.24 port 3587 ssh2</td>
</tr>
</tbody>
</table>

You extend the search using the `first` function.

```
sourcetype=secure invalid user "sshd[5258]" | table _time source _raw | stats first(_raw)
```

The search returns the event with the `_time` value 2018-04-28 00:15:05, which is the first event in the list of values returned.

<table>
<thead>
<tr>
<th>_time</th>
<th>source</th>
<th>_raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-28 00:15:05</td>
<td>tutorialdata.zip:/mailsv/secure.log</td>
<td>Mon Apr 28 2018 00:15:05 mailsv1 sshd[5258]: Failed password for invalid user tomcat from 67.170.226.218 port 1490 ssh2</td>
</tr>
</tbody>
</table>

**last(X)**

**Description**

Returns the last seen value of the field X. In general, the last seen value of the field is the oldest instance of this field relative to the input order of events into the `stats` command.
**Usage**

- To locate the last value based on time order, use the `latest` function instead.
- Works best when the search includes the `sort` command immediately before the statistics or charting command.
- This function processes field values as strings.

You can use the `last(X)` function with the `chart`, `stats`, and `timechart` commands.

**Basic example**

The following example returns the first "log_level" value for each distinct "sourcetype".

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range **All time** when you run the search.

You run the following search to locate invalid user login attempts against a specific sshd (Secure Shell Daemon). You use the table command to see the values in the `_time`, `source`, and `_raw` fields.

```
sourcetype=secure invalid user "sshd[5258]" | table _time source _raw
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>source</th>
<th>_raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-28</td>
<td>tutorialdata.zip::/mailsv/secure.log</td>
<td>Mon Apr 28 2018 00:15:05 mailsv1 sshd[5258]: Failed password for invalid user tomcat from 67.170.226.218 port 1490 ssh2</td>
</tr>
<tr>
<td>2018-05-01</td>
<td>tutorialdata.zip::/www2/secure.log</td>
<td>Thu May 01 2018 00:15:04 www2 sshd[5258]: Failed password for invalid user brian from 130.253.37.97 port 4284 ssh2</td>
</tr>
<tr>
<td></td>
<td>tutorialdata.zip::/www3/secure.log</td>
<td></td>
</tr>
</tbody>
</table>
You extend the search using the `last` function.

```
sourcetype=secure invalid user "sshd[5258]" | table _time source _raw | stats last(_raw)
```

The search returns the event with the `_time` value `2018-04-27 00:15:01`, which is the last event in the list of events. However it is not the last chronological event.
Multivalue stats and chart functions

list(X)

Description

Returns a list of up to 100 values of the field X as a multivalue entry. The order of the values reflects the order of input events.

Usage

• If more than 100 values are in field X, only the first 100 are returned.
• This function processes field values as strings.
• You can use the list(X) function with the chart, stats, and timechart commands.

Basic example

To illustrate what the list function does, let's start by generating a few simple results.

1. Use the makeresults and streamstats commands to generate a set of results that are simply timestamps and a count of the results which are used as row numbers.

| makeresults count=1000 | streamstats count AS rowNumber |

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>rowNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-02 20:27:11</td>
<td>1</td>
</tr>
<tr>
<td>2018-04-02 20:27:11</td>
<td>2</td>
</tr>
<tr>
<td>2018-04-02 20:27:11</td>
<td>3</td>
</tr>
<tr>
<td>2018-04-02 20:27:11</td>
<td>4</td>
</tr>
<tr>
<td>2018-04-02 20:27:11</td>
<td>5</td>
</tr>
</tbody>
</table>

Notice that each result appears on a separate row.
2. Add the `stats` command with the `list` function to the search. The numbers are returned in ascending order in a single, multivalue result.

```
| makeresults count=1000 | streamstats count AS rowNumber | stats list(rowNumber) AS numbers
```

The results appear on the Statistics tab and look something like this:

```
numbers
1
2
3
4
5
```

Notice that it is a single result. There are no alternating row background colors.

3. Compare this result with the results returned by the `values` function.

**values(X)**

*Description*

Returns the list of all distinct values of the field X as a multivalue entry. The order of the values is lexicographical.

*Usage*

- By default there is no limit to the number of values returned. Users with the appropriate permissions can specify a limit in the `limits.conf` file. You specify the limit in the `[stats | sistats]` stanza using the `maxvalues` setting.
- This function processes field values as strings.
- You can use the `values(X)` function with the `chart`, `stats`, and `timechart` commands.

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.

- Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically.
as 10, 100, 70, 9.
- Uppercase letters are sorted before lowercase letters.
- Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

**Basic example**

To illustrate what the `values` function does, let's start by generating a few simple results.

1. Use the `makeresults` and `streamstats` commands to generate a set of results that are simply timestamps and a count of the results, which are used as row numbers.

   ```
   | makeresults count=1000 | streamstats count AS rowNumber
   ```

   The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>rowNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-02 20:27:11</td>
<td>1</td>
</tr>
<tr>
<td>2018-04-02 20:27:11</td>
<td>2</td>
</tr>
<tr>
<td>2018-04-02 20:27:11</td>
<td>3</td>
</tr>
<tr>
<td>2018-04-02 20:27:11</td>
<td>4</td>
</tr>
<tr>
<td>2018-04-02 20:27:11</td>
<td>5</td>
</tr>
</tbody>
</table>

   Notice that each result appears on a separate row.

2. Add the `stats` command with the `values` function to the search. The results are returned in lexicographical order.

   ```
   | makeresults count=1000 | streamstats count AS rowNumber | stats values(rowNumber) AS numbers
   ```

   The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>102</td>
</tr>
</tbody>
</table>
Notice that it is a single result. There are no alternating row background colors.

3. Compare these results with the results returned by the `list` function.

### Time functions

#### earliest(X)

**Description**

Returns the chronologically earliest seen occurrence of a value of a field X.

**Usage**

- This function processes field values as strings.
- You can use the `earliest(X)` function with the `chart`, `mstats`, `stats`, `timechart`, and `tstats` commands.

**Basic example**

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to **get the tutorial data into Splunk**. Use the time range **All time** when you run the search.

You run the following search to locate invalid user login attempts against a sshd (Secure Shell Daemon). You use the table command to see the values in the `_time`, `source`, and `_raw` fields.

```
sourcetype=secure invalid user "sshd[5258]" | table _time source _raw
```
The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>source</th>
<th>_raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-30 00:15:05</td>
<td>tutorialdata.zip::mailsv/secure.log</td>
<td>Mon Apr 28 2018 00:15:05 mailsv1 sshd[5258]: Failed password for invalid user tomcat from 67.170.226.218 port 1490 ssh2</td>
</tr>
<tr>
<td>2018-04-29 00:15:04</td>
<td>tutorialdata.zip::www2/secure.log</td>
<td>Thu May 01 2018 00:15:04 www2 sshd[5258]: Failed password for invalid user brian from 130.253.37.97 port 4284 ssh2</td>
</tr>
<tr>
<td>2018-04-29 00:15:02</td>
<td>tutorialdata.zip::www3/secure.log</td>
<td>Wed Apr 30 2018 00:15:02 www3 sshd[5258]: Failed password for invalid user operator from 222.169.224.226 port 1711 ssh2</td>
</tr>
<tr>
<td>2018-04-28 00:15:01</td>
<td>tutorialdata.zip::www1/secure.log</td>
<td>Mon Apr 28 2018 00:15:01 www1 sshd[5258]: Failed password for invalid user rightscale from 87.194.216.51 port 3361 ssh2</td>
</tr>
<tr>
<td>2018-04-28 00:15:05</td>
<td>tutorialdata.zip::mailsv/secure.log</td>
<td>Thu May 01 2018 00:15:05 mailsv1 sshd[5258]: Failed password for invalid user testuser from 194.8.74.23 port 3626 ssh2</td>
</tr>
<tr>
<td>2018-04-27 00:15:01</td>
<td>tutorialdata.zip::www1/secure.log</td>
<td>Sun Apr 27 2018 00:15:01 www1 sshd[5258]: Failed password for invalid user redmine from 91.208.184.24 port 3587 ssh2</td>
</tr>
</tbody>
</table>

You extend the search using the `earliest` function.

`sourcetype=secure invalid user "sshd[5258]" | table _time source _raw | stats earliest(_raw)`

163
The search returns the event with the _time value 2018-04-27 00:15:01, which is the event with the oldest timestamp.

<table>
<thead>
<tr>
<th>_time</th>
<th>source</th>
<th>_raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-27 00:15:01</td>
<td>tutorialdata.zip:/www1/secure.log</td>
<td>Sun Apr 27 2018 00:15:01 www1 sshd[5258]: Failed password for invalid user redmine from 91.208.184.24 port 3587 ssh2</td>
</tr>
</tbody>
</table>

earliest_time(x)

**Description**

Returns the UNIX time of the chronologically earliest-seen occurrence of a given field value.

**Usage**

- This function processes field values as strings.
- You can use the earliest_time(x) function with the stats, mstats, and tstats commands.
- If you have metrics data, you can use earliest_time(x) in conjunction with earliest(x), latest(x), and latest_time(x) to calculate the rate of increase for a counter. Alternatively you can use the rate(x) counter to do the same thing.

**Basic example**

The following search runs against metric data. It is designed to return the earliest UNIX time values on every minute for each metric_name that begins with deploy.

```
| mstats earliest_time(_value) where index=_metrics metric_name=deploy* BY metric_name span=1m
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>metric_name</th>
<th>earliest_time(_value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-11-11 18:14:00</td>
<td>deploy-connections.nCurrent</td>
<td>1541988860.000000</td>
</tr>
<tr>
<td>2018-11-11 18:14:00</td>
<td>deploy-connections.nStarted</td>
<td>1541988860.000000</td>
</tr>
</tbody>
</table>
### latest(X)

**Description**

Returns the chronologically latest seen occurrence of a value of a field X.

**Usage**

- This function processes field values as strings.
- You can use the `latest(X)` function with the `chart`, `mstats`, `stats`, `timechart`, and `tstats` commands.

**Basic example**

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

You run the following search to locate invalid user login attempts against a specific sshd (Secure Shell Daemon). You use the table command to see the values in the `_time`, `source`, and `_raw` fields.

```
sourcetype=secure invalid user "sshd[5258]" | table _time source _raw
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>source</th>
<th>_raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-28</td>
<td>tutorialdata.zip::mailsv/secure.log</td>
<td>Mon Apr 28 2018 00:15:05 mailsv1 sshd[5258]: Failed</td>
</tr>
<tr>
<td>_time</td>
<td>source</td>
<td>_raw</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>2018-05-01 00:15:04</td>
<td>tutorialdata.zip:/www2/secure.log</td>
<td>Thu May 01 2018 00:15:04 www2 sshd[5258]: Failed password for invalid user brian from 130.253.37.97 port 4284 ssh2</td>
</tr>
<tr>
<td>2018-04-30 00:15:02</td>
<td>tutorialdata.zip:/www3/secure.log</td>
<td>Wed Apr 30 2018 00:15:02 www3 sshd[5258]: Failed password for invalid user operator from 222.169.224.226 port 1711 ssh2</td>
</tr>
<tr>
<td>2018-04-28 00:15:01</td>
<td>tutorialdata.zip:/www1/secure.log</td>
<td>Mon Apr 28 2018 00:15:01 www1 sshd[5258]: Failed password for invalid user rightscale from 87.194.216.51 port 3361 ssh2</td>
</tr>
<tr>
<td>2018-05-01 00:15:05</td>
<td>tutorialdata.zip:/mailsv/secure.log</td>
<td>Thu May 01 2018 00:15:05 mailsv1 sshd[5258]: Failed password for invalid user testuser from 194.8.74.23 port 3626 ssh2</td>
</tr>
<tr>
<td>2018-04-27 00:15:01</td>
<td>tutorialdata.zip:/www1/secure.log</td>
<td>Sun Apr 27 2018 00:15:01 www1 sshd[5258]: Failed password for invalid user redmine from 91.208.184.24 port 3587 ssh2</td>
</tr>
</tbody>
</table>

You extend the search using the `latest` function.

`sourcetype=secure invalid user "sshd[5258]" | table _time source _raw | stats latest(_raw)`

The search returns the event with the _time value `2018-05-01 00:15:05`, which is the event with the most recent timestamp.
latest_time(x)

Description

Returns the UNIX time of the chronologically latest-seen occurrence of a given field value.

Usage

- This function processes field values as strings.
- You can use the latest_time(x) function with the stats, mstats, and tstats commands.
- If you have metrics data, you can use latest_time(x) in conjunction with earliest(x), latest(x), and earliest_time(x) to calculate the rate of increase for a counter. Alternatively, you can use the rate(x) counter to do the same thing.

Basic example

The following search runs against metric data. It is designed to return the earliest UNIX time values in the past 60 minutes for metrics with names that begin with deploy.

| mstats latest_time(_value) where index=_metrics metric_name=queue.* BY metric_name span=1m

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>metric_name</th>
<th>earliest_time(_value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-11-13 14:43:00</td>
<td>queue.current_size</td>
<td>1542149039.000000</td>
</tr>
<tr>
<td>2018-11-13 14:43:00</td>
<td>queue.current_size_kb</td>
<td>1542149039.000000</td>
</tr>
<tr>
<td>2018-11-13 14:43:00</td>
<td>queue.largest_size</td>
<td>1542149039.000000</td>
</tr>
<tr>
<td>2018-11-13 14:43:00</td>
<td>queue.max_size_kb</td>
<td>1542149039.000000</td>
</tr>
<tr>
<td>2018-11-13 14:43:00</td>
<td>queue.smallest_size</td>
<td>1542149039.000000</td>
</tr>
<tr>
<td>_time</td>
<td>metric_name</td>
<td>earliest_time(_value)</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>2018-11-13 14:44:00</td>
<td>queue.current_size</td>
<td>1542149070.000000</td>
</tr>
<tr>
<td>2018-11-13 14:44:00</td>
<td>queue.current_size_kb</td>
<td>1542149070.000000</td>
</tr>
<tr>
<td>2018-11-13 14:44:00</td>
<td>queue.largest_size</td>
<td>1542149070.000000</td>
</tr>
<tr>
<td>2018-11-13 14:44:00</td>
<td>queue.max_size_kb</td>
<td>1542149070.000000</td>
</tr>
<tr>
<td>2018-11-13 14:44:00</td>
<td>queue.smallest_size</td>
<td>1542149070.000000</td>
</tr>
</tbody>
</table>

**per_day(X)**

*Description*

Returns the values of field X, or eval expression X, for each day.

*Usage*

- You can use the `per_day(X)` function with the `timechart` command.

**Basic examples**

The following example returns the values for the field `total` for each day.

```plaintext
... | timechart per_day(total)
```

The following example returns the results of the eval expression `eval(method="GET")] AS Views.

```plaintext
... | timechart per_day(eval(method="GET")] AS Views
```

**Extended example**

This example uses the sample dataset from the Search Tutorial but should work with any format of Apache Web access log. Download the data set from this [topic in the Search Tutorial](#) and follow the instructions to upload it to your Splunk deployment.

This search uses the `per_day()` function and eval expressions to determine how many times the web pages were viewed and how many times items were purchased. The results appear on the Statistics tab.

```plaintext
sourcetype=access_* | timechart per_day(eval(method="GET")] AS Views_day, per_day(eval(action="purchase")] AS Purchases
```
To determine the number of Views and Purchases for each hour, minute, or second you can add the other time functions to the search. For example:

```
sourceType=access_* | timechart per_day(eval(method="GET")) AS Views_day, per_hour(eval(method="GET")) AS Views_hour, per_minute(eval(method="GET")) AS Views_minute, per_day(eval(action="purchase")) AS Purchases
```

Use the field format option to change the number formatting for the field values.

**per_hour(X)**

**Description**

Returns the values of field X, or eval expression X, for each hour.

**Usage**

- You can use the `per_hour(X)` function with the `timechart` command.

**Basic examples**

The following example returns the values for the field `total` for each hour.

```
... | timechart per_hour(total)
```

The following example returns the the results of the eval expression `eval(method="POST")` AS Views.

```
... | timechart per_hour(eval(method="POST")) AS Views
```
per_minute(X)

Description

Returns the values of field X, or eval expression X, for each minute.

Usage

- You can use the `per_minute(X)` function with the `timechart` command.

Basic examples

The following example returns the values for the field `total` for each minute.

```plaintext
... | timechart per_minute(total)
```

The following example returns the results of the eval expression `eval(method="GET")] AS Views`.

```plaintext
... | timechart per_minute(eval(method="GET")) AS Views
```

per_second(X)

Description

Returns the values of field X, or eval expression X, for each second.

Usage

- You can use the `per_second(X)` function with the `timechart` command.

Basic examples

The following example returns the values for the field `kb` for each second.

```plaintext
... | timechart per_second(kb)
```

rate(x)

Description

Returns the per-second rate change of the value of the field. Represents `(latest(X) - earliest(X)) / (latest_time(X) - earliest_time(X))`. It also handles
the largest value reset if there is at least one reset.

**Usage**

- You can use the `rate(X)` function with the `stats`, `mstats`, and `tstats` commands.
- Provides the per-second rate change for accumulating counter metrics. Accumulating counters report the total counter value since the last counter reset.
- Requires the `earliest(X)` and `latest(X)` values of the field to be numerical, and the `earliest_time(X)` and `latest_time(X)` values to be different.
- Requires at least two metrics data points in the search time range.
- Should be used to provide rate information about single, rather than multiple, counters.

**Basic example**

The following search runs against metric data. It provides the hourly hit rate for a metric that provides measurements of incoming web traffic. It uses the `processor` filter to ensure that it is not reporting on multiple metric series (name and processor combinations).

```bash
|mstats rate(traffic.incoming) as rate_hits where index=_metrics
name=indexerpipe processor=index_thruput span=1h
```

The resulting chart shows you that the counter hit rate for the `traffic.incoming` metric spiked at 1 pm, 4 pm, and 11 am, but otherwise remained stable.
**Time Format Variables and Modifiers**

**Date and time format variables**

This topic lists the variables that you can use to define time formats in the evaluation functions, `strftime()` and `strptime()`. You can also use these variables to describe timestamps in event data.

Additionally, you can use the `relative_time()` and `now()` time functions as arguments.

For more information about working with dates and time, see Time modifiers for search and About searching with time in the Search Manual.

Refer to the list of tz database time zones for all permissible time zone values. For more information about how the Splunk software determines a time zone and the tz database, see Specify time zones for timestamps in Getting Data In.

**Date and time variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%c</td>
<td>The date and time in the current locale's format as defined by the server's operating system. For example, Mon Jul 13 09:30:00 2017 for US English on Linux.</td>
</tr>
<tr>
<td>%+</td>
<td>The date and time with time zone in the current locale's format as defined by the server's operating system. For example, Mon Jul 13 09:30:00 PDT 2017 for US English on Linux.</td>
</tr>
</tbody>
</table>

**Time variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Ez</td>
<td>Splunk-specific, timezone in minutes.</td>
</tr>
<tr>
<td>%H</td>
<td>Hour (24-hour clock) as a decimal number. Hours are represented by the values 00 to 23. Leading zeros are accepted but not required.</td>
</tr>
<tr>
<td>%l</td>
<td>Hour (12-hour clock) with the hours represented by the values 01 to 12. Leading zeros are accepted but not required.</td>
</tr>
<tr>
<td>%k</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Like %H, the hour (24-hour clock) as a decimal number. Leading zeros are replaced by a space, for example 0 to 23.</td>
<td>%M</td>
</tr>
<tr>
<td>Minute as a decimal number. Minutes are represented by the values 00 to 59. Leading zeros are accepted but not required.</td>
<td>%N</td>
</tr>
<tr>
<td>The number of subsecond digits. The default is %9N. You can specify %3N = milliseconds, %6N = microseconds, %9N = nanoseconds.</td>
<td>%p</td>
</tr>
<tr>
<td>AM or PM.</td>
<td>%Q</td>
</tr>
<tr>
<td>The subsecond component of a UTC timestamp. The default is milliseconds, %3Q. Valid values are:</td>
<td>%S</td>
</tr>
<tr>
<td>• %3Q = milliseconds, with values of 000-999</td>
<td>%s</td>
</tr>
<tr>
<td>• %6Q = microseconds, with values of 000000-999999</td>
<td>The Unix Epoch Time timestamp, or the number of seconds since the Epoch: 1970-01-01 00:00:00 +0000 (UTC). (1484993700 is Sat Jan 21 10:15:00 2017)</td>
</tr>
<tr>
<td>• %9Q = nanoseconds, with values of 000000000-999999999</td>
<td>The time in 24-hour notation (%H:%M:%S). For example 23:59:59.</td>
</tr>
<tr>
<td>Second as a decimal number, for example 00 to 59.</td>
<td>The time in the format for the current locale. For US English the format for 9:30 AM is 9:30:00.</td>
</tr>
<tr>
<td>The timezone abbreviation. For example EST for US Eastern Standard Time.</td>
<td>The timezone offset from UTC, in hour and minute: +hhmm or -hhmm. For example, for 5 hours before UTC the values is -0500 which is US Eastern Standard Time.</td>
</tr>
</tbody>
</table>

Examples:

- Use %z to specify hour and minute, for example -0500
- Use %:z to specify hour and minute separated by a colon, for example -5:00
- Use %::z to specify hour minute and second separated with colons, for example -05:00:00
- Use %:::z to specify hour only, for example -05
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%%</td>
<td>A literal &quot;%&quot; character.</td>
</tr>
</tbody>
</table>

**Date variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%F</td>
<td>Equivalent to %Y-%m-%d (the ISO 8601 date format).</td>
</tr>
<tr>
<td>%x</td>
<td>The date in the format of the current locale. For example, 7/13/2017 for US English.</td>
</tr>
</tbody>
</table>

**Specifying days and weeks**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%A</td>
<td>Full weekday name. (Sunday, ..., Saturday)</td>
</tr>
<tr>
<td>%a</td>
<td>Abbreviated weekday name. (Sun, ... ,Sat)</td>
</tr>
<tr>
<td>%d</td>
<td>Day of the month as a decimal number, includes a leading zero. (01 to 31)</td>
</tr>
<tr>
<td>%e</td>
<td>Like %d, the day of the month as a decimal number, but a leading zero is replaced by a space. (1 to 31)</td>
</tr>
<tr>
<td>%j</td>
<td>Day of year as a decimal number, includes a leading zero. (001 to 366)</td>
</tr>
<tr>
<td>%V</td>
<td>Week of the year. (1 to 52)</td>
</tr>
<tr>
<td>%w</td>
<td>Weekday as a decimal number. (0 = Sunday, ..., 6 = Saturday)</td>
</tr>
</tbody>
</table>

**Specifying months**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%b</td>
<td>Abbreviated month name. (Jan, Feb, etc.)</td>
</tr>
<tr>
<td>%B</td>
<td>Full month name. (January, February, etc.)</td>
</tr>
<tr>
<td>%m</td>
<td>Month as a decimal number. (01 to 12). Leading zeros are accepted but not required.</td>
</tr>
</tbody>
</table>

**Specifying year**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%y</td>
<td>Year as a decimal number, without the century. (00 to 99). Leading zeros are accepted but not required.</td>
</tr>
<tr>
<td>%Y</td>
<td>Year as a decimal number with century. For example, 2017.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Examples</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time format string</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Y-%m-%d</td>
<td>2017-12-31</td>
</tr>
<tr>
<td>%y-%m-%d</td>
<td>17-12-31</td>
</tr>
<tr>
<td>%b %d, %Y</td>
<td>Feb 11, 2017</td>
</tr>
<tr>
<td>q</td>
<td>%d%b 'yy = %Y-%m-%d</td>
</tr>
<tr>
<td>host=&quot;www1&quot;</td>
<td>eval WeekNo = strftime(_time,&quot;%V&quot;)</td>
</tr>
<tr>
<td>...</td>
<td>eval mytime=strftime(_time,&quot;%Y-%m-%dT%H:%M:%S.%Q&quot;)</td>
</tr>
</tbody>
</table>

**Time modifiers**

Use time modifiers to customize the time range of a search or change the format of the timestamps in the search results.

**_time and _indextime fields**

When an event is processed by Splunk software, its timestamp is saved as the default field _time. This timestamp, which is the time when the event occurred, is saved in UNIX time notation. Searching with relative time modifiers, earliest or latest, finds every event with a timestamp beginning, ending, or between the specified timestamps.

For example, when you search for earliest=@d, the search finds every event with a _time value since midnight. This example uses @d, which is a date format variable. See Date and time format variables.

You also have the option of searching for events based on when they were indexed. The UNIX time is saved in the _indextime field. Similar to earliest and latest for the _time field, you can use the relative time modifiers _index_earliest and _index_latest to search for events based on _indextime. For example, if you wanted to search for events indexed in the previous hour,
Note: When using index-time based modifiers such as \_index\_earliest and \_index\_latest, your search must also have an event-time window which will retrieve the events. In other words, chunks of events might be ruled out based on the non index-time window as well as the index-time window. To be certain of retrieving every event based on index-time, you must run your search using All Time.

List of time modifiers

Use the earliest and latest modifiers to specify custom and relative time ranges. You can specify an exact time such as earliest="10/5/2016:20:00:00", or a relative time such as earliest=-h or latest=@w6.

When specifying relative time, you can use the now modifier to refer to the current time.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>earliest</td>
<td>earliest=[+-]</td>
<td>Specify the earliest _time for the time range of your search.</td>
</tr>
<tr>
<td></td>
<td>&lt;time_integer&gt;&lt;time_unit&gt;@&lt;time_unit&gt;</td>
<td>Use earliest=1 to specify the UNIX epoch time 1, which is UTC January 1, 1970 at 12:00:01 AM.</td>
</tr>
<tr>
<td>_index_earliest</td>
<td>_index_earliest=[+-]</td>
<td>Specify the earliest _indextime for the time range of your search.</td>
</tr>
<tr>
<td></td>
<td>&lt;time_integer&gt;&lt;time_unit&gt;@&lt;time_unit&gt;</td>
<td></td>
</tr>
<tr>
<td>_index_latest</td>
<td>_index_latest=[+-]</td>
<td>Specify the latest _indextime for the time range of your search.</td>
</tr>
<tr>
<td></td>
<td>&lt;time_integer&gt;&lt;time_unit&gt;@&lt;time_unit&gt;</td>
<td></td>
</tr>
<tr>
<td>latest</td>
<td>latest=[+-]</td>
<td>Specify the latest _time for the _time</td>
</tr>
</tbody>
</table>
How to specify relative time modifiers

You can define the relative time in your search with a string of characters that indicate time amount (integer and unit). You can also specify a "snap to" time unit, which is specified with the @ symbol followed by a time unit.

The syntax for using time modifiers is

\[ [+|-] <time_integer><time_unit>@<time_unit> \]

The steps to specify a relative time modifier are:

1. Indicate the time offset from the current time.
2. Define the time amount, which is a number and a unit.
3. Specify a "snap to" time unit. The time unit indicates the nearest or latest time to which your time amount rounds down.

**Indicate the time offset**

Begin your string with a plus (+) or minus (-) to indicate the offset from the current time.

**Define the time amount**

Define your time amount with a number and a unit. The supported time units are listed in the following table.
<table>
<thead>
<tr>
<th>Time unit</th>
<th>Valid unit abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>second</td>
<td>s, sec, secs, second, seconds</td>
</tr>
<tr>
<td>minute</td>
<td>m, min, minute, minutes</td>
</tr>
<tr>
<td>hour</td>
<td>h, hr, hrs, hour, hours</td>
</tr>
<tr>
<td>day</td>
<td>d, day, days</td>
</tr>
<tr>
<td>week</td>
<td>w, week, weeks</td>
</tr>
<tr>
<td>month</td>
<td>mon, month, months</td>
</tr>
<tr>
<td>quarter</td>
<td>q, qtr, qtrs, quarter, quarters</td>
</tr>
<tr>
<td>year</td>
<td>y, yr, yrs, year, years</td>
</tr>
</tbody>
</table>

For example, to start your search an hour ago, use either of the following time modifiers.

`earliest=-h`

or

`earliest=-60m`

When specifying single time amounts, the number one is implied. An 's' is the same as '1s', 'm' is the same as '1m', 'h' is the same as '1h', and so forth.

**Specify a snap to time unit**

You can specify a snap to time unit. The time unit indicates the nearest or latest time to which your time amount rounds down. Separate the time amount from the "snap to" time unit with an "@" character.

- You can use any of time units listed previously. For example:
  - @w, @week, and @w0 for Sunday
  - @month for the beginning of the month
  - @q, @qtr, or @quarter for the beginning of the most recent quarter (Jan 1, Apr 1, Jul 1, or Oct 1).
- You can specify a day of the week: w0 (Sunday), w1, w2, w3, w4, w5 and w6 (Saturday). For Sunday, you can specify w0 or w7.
- You can also specify offsets from the snap-to-time or "chain" together the time modifiers for more specific relative time definitions. For example, @d-2h snaps to the beginning of today (12 AM or midnight), and then applies the time offset of -2h, This results in a time of 10 PM yesterday.
  - The Splunk platform always applies the offset before it applies the snap. In other words, the left-hand side of the @ symbol is applied
before the right-hand side.

- When snapping to the nearest or latest time, Splunk software always **snaps backwards** or rounds down to the latest time not after the specified time. For example, if it is 11:59:00 and you "snap to" hours, you will snap to 11:00 not 12:00.
- If you do not specify a time offset before the "snap to" amount, Splunk software interprets the time as "current time snapped to" the specified amount. For example, if it is currently 11:59 PM on Friday and you use `@w6` to "snap to Saturday", the resulting time is the **previous** Saturday at 12:01 A.M.

**Examples**

1. **Run a search over all time**

If you want to search events from the start of UNIX time, use `earliest=1`.

When `earliest=1` and `latest=now()` are used, the search runs over all time.

```plaintext
...earliest=1 latest=now()
```

Specifying `latest=now()` does not return future events.

To return future events, specify `latest=<a big number>`. Future events are events that contain timestamps later than the current time `now()`.

2. **Search the events from the beginning of the current week**

`earliest=@w0`

3. **Search the events from the last full business week**

`earliest=-5d@w1 latest=@w6`

4. **Search with an exact date as a boundary**

With a boundary such as from November 15 at 8 PM to November 22 at 8 PM, use the `timeformat` `%m/%d/%Y:%H:%M:%S`.
earliest="11/15/2017:20:00:00" latest="11/22/2017:20:00:00"

5. Specify multiple time windows

You can specify multiple time windows using the timeformat %m/%d/%Y:%H:%M:%S. For example to find events from 5-6 PM or 7-8 PM on specific dates, use the following syntax.

(earliest="1/22/2018:17:00:00" latest="1/22/2018:18:00:00") OR
(earliest="1/22/2018:19:00:00" latest="1/22/2018:20:00:00")

Other time modifiers

These search time modifiers are still valid, but might be removed and their function no longer supported in a future release.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>daysago</td>
<td>daysago=&lt;int&gt;</td>
<td>Search events within the last integer number of days.</td>
</tr>
<tr>
<td>enddaysago</td>
<td>enddaysago=&lt;int&gt;</td>
<td>Set an end time for an integer number of days before Now.</td>
</tr>
<tr>
<td>endhoursago</td>
<td>endhoursago=&lt;int&gt;</td>
<td>Set an end time for an integer number of hours before Now.</td>
</tr>
<tr>
<td>endminutesago</td>
<td>endminutesago=&lt;int&gt;</td>
<td>Set an end time for an integer number of minutes before Now.</td>
</tr>
<tr>
<td>endmonthsago</td>
<td>endmonthsago=&lt;int&gt;</td>
<td>Set an end time for an integer number of months before Now.</td>
</tr>
<tr>
<td>endtime</td>
<td>endtime=&lt;string&gt;</td>
<td>Search for events before the specified time (exclusive of the specified time). Use timeformat to specify how the timestamp is formatted.</td>
</tr>
<tr>
<td>endtimeu</td>
<td>endtimeu=&lt;int&gt;</td>
<td>Search for events before the specific UNIX time.</td>
</tr>
<tr>
<td>hoursago</td>
<td>hoursago=&lt;int&gt;</td>
<td>Search events within the last integer number of hours.</td>
</tr>
<tr>
<td>minutesago</td>
<td>minutesago=&lt;int&gt;</td>
<td>Search events within the last integer number of minutes.</td>
</tr>
<tr>
<td>monthsago</td>
<td>monthsago=&lt;int&gt;</td>
<td></td>
</tr>
<tr>
<td>Modifier</td>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>searchtimespandays</td>
<td>searchtimespandays=&lt;int&gt;</td>
<td>Search within a specified range of days, expressed as an integer.</td>
</tr>
<tr>
<td>searchtimespanhours</td>
<td>searchtimespanhours=&lt;int&gt;</td>
<td>Search within a specified range of hours, expressed as an integer.</td>
</tr>
<tr>
<td>searchtimespanminutes</td>
<td>searchtimespanminutes=&lt;int&gt;</td>
<td>Search within a specified range of minutes, expressed as an integer.</td>
</tr>
<tr>
<td>searchtimespanmonths</td>
<td>searchtimespanmonths=&lt;int&gt;</td>
<td>Search within a specified range of months, expressed as an integer.</td>
</tr>
<tr>
<td>startdaysago</td>
<td>startdaysago=&lt;int&gt;</td>
<td>Search the specified number of days before the present time.</td>
</tr>
<tr>
<td>starthoursago</td>
<td>starthoursago=&lt;int&gt;</td>
<td>Search the specified number of hours before the present time.</td>
</tr>
<tr>
<td>startminutesago</td>
<td>startminutesago=&lt;int&gt;</td>
<td>Search the specified number of minutes before the present time.</td>
</tr>
<tr>
<td>startmonthsago</td>
<td>startmonthsago=&lt;int&gt;</td>
<td>Search the specified number of months before the present time.</td>
</tr>
<tr>
<td>starttime</td>
<td>starttime=&lt;timestamp&gt;</td>
<td>Search from the specified date and time to the present, inclusive of the specified time.</td>
</tr>
<tr>
<td>starttimeu</td>
<td>starttimeu=&lt;int&gt;</td>
<td>Search for events starting from the specific UNIX time.</td>
</tr>
<tr>
<td>timeformat</td>
<td>timeformat=&lt;string&gt;</td>
<td>Set the timeformat for the <strong>starttime</strong> and <strong>endtime</strong> modifiers. By default: timeformat=%m/%d/%Y:%H:%M:%S</td>
</tr>
</tbody>
</table>
Search Commands

abstract

Description

Produces an abstract, a summary or brief representation, of the text of the search results. The original text is replaced by the summary.

The abstract is produced by a scoring mechanism. Events that are larger than the selected maxlines, those with more textual terms and more terms on adjacent lines, are preferred over events with fewer terms. If a line has a search term, its neighboring lines also partially match, and might be returned to provide context. When there are gaps between the selected lines, lines are prefixed with an ellipsis (...).

If the text of an event has fewer lines or an equal number of lines as maxlines, no change occurs.

Syntax

abstract [maxterms=<int>] [maxlines=<int>]

Optional arguments

maxterms

Syntax: maxterms=<int>
Description: The maximum number of terms to match. Accepted values are 1 to 1000.

maxlines

Syntax: maxlines=<int>
Description: The maximum number of lines to match. Accepted values are 1 to 500.

Examples

Example 1: Show a summary of up to 5 lines for each search result.

... |abstract maxlines=5
See also

highlight

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the abstract command.

accum

Description

For each event where field is a number, the accum command calculates a running total or sum of the numbers. The accumulated sum can be returned to either the same field, or a newfield that you specify.

Syntax

accum <field> [AS <newfield>]

Required arguments

field

Syntax: <string>
Description: The name of the field that you want to calculate the accumulated sum for. The field must contain numeric values.

Optional arguments

newfield

Syntax: <string>
Description: The name of a new field where you want the results placed.

Examples

Example 1:

Save the running total of the quantity field into a new field called "total_quantity".

... | accum quantity AS total_quantity
See also

autoregress, delta, streamstats, trendline

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the accum command.

addcoltotals

Description

The `addcoltotals` command appends a new result to the end of the search result set. The result contains the sum of each numeric field or you can specify which fields to summarize. Results are displayed on the Statistics tab. If the `labelfield` argument is specified, a column is added to the statistical results table with the name specified.

Syntax

```
addcoltotals [labelfield=<field>] [label=<string>] [<fieldlist>]
```

Optional arguments

- `<fieldlist>`
  - Syntax: `<field>` ...
  - Description: A space delimited list of valid field names. The `addcoltotals` command calculates the sum only for the fields in the list you specify. You can use the asterisk ( * ) as a wildcard in the field names.
  - Default: Calculates the sum for all of the fields.

- `labelfield`
  - Syntax: `labelfield=<fieldname>`
  - Description: Specify a field name to add to the result set.
  - Default: none

- `label`
  - Syntax: `label=<string>`
  - Description: Used with the `labelfield` argument to add a label in the summary event. If the `labelfield` argument is absent, the `label` argument
has no effect.

**Default:** Total

**Examples**

**Example 1:**

Compute the sums of all the fields, and put the sums in a summary event called "change_name".

```bash
... | addcoltotals labelfield=change_name label=ALL
```

**Example 2:**

Add a column total for two specific fields in a table.

```bash
sourcetype=access_* | table userId bytes avgTime duration | addcoltotals bytes duration
```

**Example 3:**

Filter fields for two name-patterns, and get totals for one of them.

```bash
... | fields user*, *size | addcoltotals *size
```

**Example 4:**

Augment a chart with a total of the values present.

```bash
index=_internal source="metrics.log" group=pipeline |stats avg(cpu_seconds) by processor |addcoltotals labelfield=processor
```

**See also**

addcoltotals, stats

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the addcoltotals command.
addinfo

**Description**

Adds fields to each event that contain global, common information about the search. This command is primarily an internally-used component of Summary Indexing.

**Syntax**

```
addinfo
```

The following fields are added to each event when you use the `addinfo` command.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info_min_time</td>
<td>The earliest time boundary for the search.</td>
</tr>
<tr>
<td>info_max_time</td>
<td>The latest time boundary for the search.</td>
</tr>
<tr>
<td>info_sid</td>
<td>The ID of the search that generated the event.</td>
</tr>
<tr>
<td>info_search_time</td>
<td>The time when the search was run.</td>
</tr>
</tbody>
</table>

**Usage**

The `addinfo` command is a **distributable streaming command**. See [Command types](#).

**Examples**

1. **Add information to each event**

Add information about the search to each event.

```bash
... | addinfo
```

2. **Determine which heartbeats are later than expected**

You can use this example to track heartbeats from hosts, forwarders, `tcpin_connections` on indexers, or any number of system components. This example uses hosts.
You have a list of host names in a lookup file called `expected_hosts`. You want to search for heartbeats from your hosts that are after an expected time range. You use the `addinfo` command to add information to each event that will help you evaluate the time range.

```
... | stats latest(_time) AS latest_time BY host | addinfo | eval latest_age = info_max_time - latest_time | fields - info_* | inputlookup append=t expected_hosts | fillnull value=9999 latest_age | dedup host | where latest_age > 42
```

Use the `stats` command to calculate the latest heartbeat by host. The `addinfo` command adds information to each result. This search uses `info_max_time`, which is the latest time boundary for the search. The `eval` command is used to create a field called `latest_age` and calculate the age of the heartbeats relative to end of the time range. This allows for a time range of `-11m@m` to `-m@m`. This is the previous 11 minutes, starting at the beginning of the minute, to the previous 1 minute, starting at the beginning of the minute. The search does not work if you specify `latest=null / all time` because `info_max_time` would be set to `+infinity`.

Using the lookup file, `expected_hosts`, append the list of hosts to the results. Using this list you can determine which hosts are not sending a heartbeat in the expected time range. For any hosts that have a null value in the `latest_age` field, fill the field with the value 9999. Remove any duplicated host events with the `dedup` command. Use the `where` command to filter the results and return any heartbeats older than 42 seconds.

In this example, you could use the `tstats` command, instead of the `stats` command, to improve the performance of the search.

**See also**

search

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `addinfo` command.

`addtotals`
Description

The addtotals command computes the arithmetic sum of all numeric fields for each search result. The results appear in the Statistics tab.

You can specify a list of fields that you want the sum for, instead of calculating every numeric field. The sum is placed in a new field.

If col=true, the addtotals command computes the column totals, which adds a new result at the end that represents the sum of each field. labelfield, if specified, is a field that will be added to this summary event with the value set by the 'label' option. Alternately, instead of using the addtotals col=true command, you can use the addcoltotals command to calculate a summary event.

Syntax

addtotals [row=<bool>] [col=<bool>] [labelfield=<field>] [label=<string>] [fieldname=<field>] [<field-list>]

Required arguments

None.

Optional arguments

field-list

Syntax: <field> ...
Description: One or more numeric fields, delimited with a space. Only the fields specified in the <field-list> are summed. If a <field-list> is not specified, all numeric fields are included in the sum.
Usage: You can use wildcards in the field names. For example, if the field names are count1, count2, and count3 you can specify count* to indicate all fields that begin with 'count'.
Default: All numeric fields are included in the sum.

row

Syntax: row=<bool>
Description: Specifies whether to calculate the sum of the <field-list> for each event. This is similar to calculating a total for each row in a table. The sum is placed in a new field. The default name of the field is Total. If you want to specify a different name for the field, use the fieldname argument.
**Usage:** Because the default is `row=true`, specify the `row` argument only when you do not want the event totals to appear `row=false`.

**Default:** true

**col**

**Syntax:** `col=<bool>`

**Description:** Specifies whether to add a new event, referred to as a summary event, at the bottom of the list of events. The summary event displays the sum of each field in the events, similar to calculating column totals in a table.

**Default:** false

**fieldname**

**Syntax:** `fieldname=<field>`

**Description:** Used to specify the name of the field that contains the calculated sum of the `field-list` for each event. The `fieldname` argument is valid only when `row=true`.

**Default:** Total

**labelfield**

**Syntax:** `labelfield=<field>`

**Description:** Used to specify a field for the summary event label. The `labelfield` argument is valid only when `col=true`.

* To use an existing field in your result set, specify the field name for the `labelfield` argument. For example if the field name is `IP`, specify `labelfield=IP`.
* If there is no field in your result set that matches the `labelfield`, a new field is added using the `labelfield` value.

**Default:** none

**label**

**Syntax:** `label=<string>`

**Description:** Used to specify a row label for the summary event.

* If the `labelfield` argument is an existing field in your result set, the label value appears in that row in the display.
* If the `labelfield` argument creates a new field, the label appears in the new field in the summary event row.

**Default:** Total

**Usage**

The `addtotals` command is a **distributable streaming command**, except when is used to calculate column totals. When used to calculate column totals, the
addtotals command is a **transforming command**. See Command types.

**Examples**

1. **Calculate the sum of the numeric fields of each event**

This example uses events that list the numeric sales for each product and quarter, for example:

<table>
<thead>
<tr>
<th>products</th>
<th>quarter</th>
<th>sales</th>
<th>quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
<td>QTR1</td>
<td>1200</td>
<td>1000</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR1</td>
<td>1400</td>
<td>1550</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR1</td>
<td>1650</td>
<td>1275</td>
</tr>
<tr>
<td>ProductA</td>
<td>QTR2</td>
<td>1425</td>
<td>1300</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR2</td>
<td>1175</td>
<td>1425</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR2</td>
<td>1575</td>
<td>1425</td>
</tr>
<tr>
<td>ProductA</td>
<td>QTR3</td>
<td>1300</td>
<td>1400</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR3</td>
<td>1250</td>
<td>1125</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR3</td>
<td>1375</td>
<td>1475</td>
</tr>
<tr>
<td>ProductA</td>
<td>QTR4</td>
<td>1550</td>
<td>1300</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR4</td>
<td>1700</td>
<td>1225</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR4</td>
<td>1625</td>
<td>1350</td>
</tr>
</tbody>
</table>

**Use the chart command to summarize data**

To summarize the data by product for each quarter, run this search:

```plaintext
source="addtotalsData.csv" | chart sum(sales) BY products quarter
```

In this example, there are two fields specified in the BY clause with the chart command.

- The products field is referred to as the <row-split> field.
- The quarter field is referred to as the <column-split> field.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>products</th>
<th>QTR1</th>
<th>QTR2</th>
<th>QTR3</th>
<th>QTR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
<td>1200</td>
<td>1425</td>
<td>1300</td>
<td>1550</td>
</tr>
</tbody>
</table>
To add a column that generates totals for each row, run this search:

```
source="addtotalsData.csv" | chart sum(sales) BY products quarter | addtotals
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>products</th>
<th>QTR1</th>
<th>QTR2</th>
<th>QTR3</th>
<th>QTR4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
<td>1200</td>
<td>1425</td>
<td>1300</td>
<td>1550</td>
<td>5475</td>
</tr>
<tr>
<td>ProductB</td>
<td>1400</td>
<td>1175</td>
<td>1250</td>
<td>1700</td>
<td>5525</td>
</tr>
<tr>
<td>ProductC</td>
<td>1650</td>
<td>1550</td>
<td>1375</td>
<td>1625</td>
<td>6200</td>
</tr>
</tbody>
</table>

### Use the stats command to calculate totals

If all you need are the totals for each product, a simpler solution is to use the `stats` command:

```
source="addtotalsData.csv" | stats sum(sales) BY products
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>products</th>
<th>sum(sales)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
<td>5475</td>
</tr>
<tr>
<td>ProductB</td>
<td>5525</td>
</tr>
<tr>
<td>ProductC</td>
<td>6200</td>
</tr>
</tbody>
</table>

### 2. Specify a name for the field that contains the sums for each event

Instead of accepting the default name added by the `addtotals` command, you can specify a name for the field.

```
... | addtotals fieldname=sum
```

### 3. Use wildcards to specify the names of the fields to sum

Calculate the sums for the fields that begin with `amount` or that contain the text `size` in the field name. Save the sums in the field called `TotalAmount`. 

<table>
<thead>
<tr>
<th>products</th>
<th>sum(sales)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
<td>5475</td>
</tr>
<tr>
<td>ProductB</td>
<td>5525</td>
</tr>
<tr>
<td>ProductC</td>
<td>6200</td>
</tr>
</tbody>
</table>
4. **Calculate the sum for a specific field**

In this example, the row calculations are turned off and the column calculations are turned on. The total for only a single field, `sum(quota)`, is calculated.

```
source="addtotalsData.csv" | stats sum(quota) by quarter| addtotals row=f col=t labelfield=quarter sum(quota)
```

- The `labelfield` argument specifies in which field the label for the total appears. The default label is **Total**.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>products</th>
<th>sum(quota)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QTR1</td>
<td>3825</td>
</tr>
<tr>
<td>QTR2</td>
<td>4175</td>
</tr>
<tr>
<td>QTR3</td>
<td>4000</td>
</tr>
<tr>
<td>QTR4</td>
<td>3875</td>
</tr>
<tr>
<td>Total</td>
<td>15875</td>
</tr>
</tbody>
</table>

5. **Calculate the field totals and add custom labels to the totals**

Calculate the sum for each quarter and product, and calculate a grand total.

```
source="addtotalsData.csv" | chart sum(sales) by products quarter| addtotals col=t labelfield=products label="Quarterly Totals" fieldname="Product Totals"
```

- The `labelfield` argument specifies in which field the label for the total appears, which in this example is **products**.
- The `label` argument is used to specify the label **Quarterly Totals** for the `labelfield`, instead of using the default label **Total**.
- The `fieldname` argument is used to specify the label **Product Totals** for the row totals.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>products</th>
<th>QTR1</th>
<th>QTR2</th>
<th>QTR3</th>
<th>QTR4</th>
<th>Product Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
<td>1200</td>
<td>1425</td>
<td>1300</td>
<td>1550</td>
<td>5475</td>
</tr>
<tr>
<td>products</td>
<td>QTR1</td>
<td>QTR2</td>
<td>QTR3</td>
<td>QTR4</td>
<td>Product Totals</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>---------------</td>
</tr>
<tr>
<td>ProductB</td>
<td>1400</td>
<td>1175</td>
<td>1250</td>
<td>1700</td>
<td>5525</td>
</tr>
<tr>
<td>ProductC</td>
<td>1650</td>
<td>1550</td>
<td>1375</td>
<td>1625</td>
<td>6200</td>
</tr>
<tr>
<td>Quarterly Totals</td>
<td>4250</td>
<td>4150</td>
<td>3925</td>
<td>4875</td>
<td>17200</td>
</tr>
</tbody>
</table>

**See also**

stats

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the addtotals command.

**analyzefields**

**Description**

Using *field* as a discrete random variable, this command analyzes all numerical fields to determine the ability for each of those fields to predict the value of the *classfield*. It determines the stability of the relationship between values in the target *classfield* and numeric values in other fields.

As a reporting command, `analyzefields` consumes all input results and generates one row for each numeric field in the output results. The values in that row indicate the performance of the `analyzefields` command at predicting the value of a *classfield*. For each event, if the conditional distribution of the numeric field with the highest z-probability based on matches the actual class, the event is counted as accurate. The highest z-probability is based on the *classfield*.

**Syntax**

`analyzefields classfield=<field>`

You can use the abbreviation `af` for the `analyzefields` command.

The `analyzefields` command returns a table with five columns.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>The name of a numeric field from the input search results.</td>
</tr>
<tr>
<td>count</td>
<td>The number of occurrences of the field in the search results.</td>
</tr>
<tr>
<td>cocur</td>
<td>The co-occurrence of the field. In the results where classfield is present, this is the ratio of results in which field is also present. The cocur is 1 if the field exists in every event that has a classfield.</td>
</tr>
<tr>
<td>acc</td>
<td>The accuracy in predicting the value of the classfield, using the value of the field. This the ratio of the number of accurate predictions to the total number of events with that field. This argument is valid only for numerical fields.</td>
</tr>
<tr>
<td>balacc</td>
<td>The balanced accuracy is the non-weighted average of the accuracies in predicted each value of the classfield. This is only valid for numerical fields.</td>
</tr>
</tbody>
</table>

**Required arguments**

classfield

**Syntax:** classfield=<field>

**Description:** For best results, classfield should have two distinct values, although multiclass analysis is possible.

**Examples**

**Example 1:**

Analyze the numerical fields to predict the value of "is_activated".

```plaintext
... | analyzefields classfield=is_activated
```

**See also**

anomalousvalue

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the analyzefields command.
anomalies

Description

Use the `anomalies` command to look for events or field values that are unusual or unexpected.

The `anomalies` command assigns an unexpectedness score to each event and places that score in a new field named `unexpectedness`. Whether the event is considered anomalous or not depends on a threshold value. The threshold value is compared to the unexpectedness score. The event is considered unexpected or anomalous if the unexpectedness score is greater than the threshold value.

After you use the `anomalies` command in a search, look at the Interesting Fields list in the Search & Reporting window. Select the `unexpectedness` field to see information about the values in your events.

The unexpectedness score of an event is calculated based on the similarity of that event (X) to a set of previous events (P).

The formula for unexpectedness is:

\[
\text{unexpectedness} = \frac{\left[ s(P \text{ and } X) - s(P) \right]}{s(P) + s(X)}
\]

In this formula, \( s( ) \) is a metric of how similar or uniform the data is. This formula provides a measure of how much adding X affects the similarity of the set of events. The formula also normalizes the results for the differing event sizes.

Syntax

```
anomalies [threshold=num] [labelonly=bool] [normalize=bool] [maxvalues=inf] [field=field] [blacklist=filename] [blacklistthreshold=num] [by-clause]
```

Optional arguments

**threshold**

**Datatype:** threshold=<num>

**Description:** A number to represent the upper limit of expected or normal events. If unexpectedness calculated for an event is greater than this threshold limit, the event is considered unexpected or anomalous.
column

Datatype: labelonly=<bool>
Description: Specifies if you want the output result set to include all events or only the events that are above the threshold value. The unexpectedness field is appended to all events. If labelonly=true, no events are removed. If labelonly=false, events that have a unexpectedness score less than the threshold are removed from the output result set.
Default: false

normalize

Datatype: normalize=<bool>
Description: Specifies whether or not to normalize numeric text in the fields. All characters in the field from 0 to 9 are considered identical for purposes of the algorithm. The placement and quantity of the numbers remains significant. When a field contains numeric data that should not be normalized but treated as categories, set normalize=false.
Default: true

maxvalues

Datatype: maxvalues=<int>
Description: Specifies the size of the sliding set of previous events to include when determining the unexpectedness of a field value. By default the calculation uses the previous 100 events for the comparison. If the current event number is 1000, the calculation uses the values in events 900 to 999 in the calculation. If the current event number is 1500, the calculation uses the values in events 1400 to 1499 in the calculation. You can specify a number between 10 and 10000. Increasing the value of maxvalues increases the total CPU cost per event linearly. Large values have very long search runtimes.
Default: 100

field

Datatype: field=<field>
Description: The field to analyze when determining the unexpectedness of an event.
Default: _raw

blacklist

Datatype: blacklist=<filename>
**Description:** The name of a CSV file that contains a list of events that are expected and should be ignored. Any incoming event that is similar to an event in the blacklist is treated as not anomalous, or expected, and given an unexpectedness score of 0.0. The CSV file must be located in the $SPLUNK_HOME/var/run/splunk/ directory on the search head. If you have Splunk Cloud and want to configure a blacklist file, file a Support ticket.

`blacklistthreshold`  
**Datatype:** `blacklistthreshold=<num>`  
**Description:** Specifies a similarity score threshold for matching incoming events to blacklisted events. If the incoming event has a similarity score above the `blacklistthreshold`, the event is marked as unexpected.  
**Default:** 0.05

**by-clause**  
**Syntax:** `by <fieldlist>`  
**Description:** Use to specify a list of fields to segregate the results for anomaly detection. For each combination of values for the specified fields, the events with those values are treated entirely separately.

**Examples**

1. **Specify a blacklist file of the events to ignore**

The following example shows the interesting events, ignoring any events in the blacklist 'boring events'. Sort the event list in descending order, with highest value in the unexpectedness field listed first.

   ```
   ... | anomalies blacklist=boringevents | sort -unexpectedness
   ```

2. **Find anomalies in transactions**

   This example uses transactions to find regions of time that look unusual.

   ```
   ... | transaction maxpause=2s | anomalies
   ```

3. **Identify anomalies by source**

   Look for anomalies in each source separately. A pattern in one source does not affect that it is anomalous in another source.

   ```
   ... | anomalies by source
   ```
4. Specify a threshold when identifying anomalies

This example shows how to tune a search for anomalies using the threshold value. Start with a search that uses the default threshold value.

```
index=_internal | anomalies BY group | search group=* 
```

This search looks at events in the `_internal` index and calculates an unexpectedness score for sets of events that have the same group value.

- The sliding set of events that are used to calculate the unexpectedness score for each unique group value includes only the events that have the same group value.
- The search command is used to show events that only include the group field.

The unexpectedness and group fields appear in the list of Interesting fields. Click on the field name and then click Yes to move the field to the Selected fields list. The fields are moved and also appear in the search results. Your results should look something like the following image.

The key-value pairs in the first event include group=pipeline, name=indexerpipe, processor=indexer, cpu_seconds=0.022, and so forth.

With the default threshold, which is 0.01, you can see that some of these events might be very similar. The next search increases the threshold a little:

```
index=_internal | anomalies threshold=0.03 by group | search group=* 
```
With the higher threshold value, the timestamps and key-value pairs show more distinction between each of the events.

Also, you might not want to hide the events that are not anomalous. Instead, you can add another field to your events that tells you whether or not the event is interesting to you. One way to do this is with the `eval` command:

```
index=_internal | anomalies threshold=0.03 labelonly=true by group |
search group=* | eval threshold=0.03 | eval score=if(unexpectedness>=threshold, "anomalous", "boring")
```

This search uses `labelonly=true` so that the boring events are still retained in the results list. The `eval` command is used to define a field named `threshold` and set it to the threshold value. This has to be done explicitly because the `threshold` attribute of the `anomalies` command is not a field.

The second `eval` command is used to define another new field, `score`, that is either "anomalous" or "boring" based on how the `unexpectedness` compares to the `threshold` value. The following image shows a snapshot of the results.
anomalousvalue

Description

The anomalousvalue command computes an anomaly score for each field of each event, relative to the values of this field across other events. For numerical fields, it identifies or summarizes the values in the data that are anomalous either by frequency of occurrence or number of standard deviations from the mean.

For fields that are determined to be anomalous, a new field is added with the following scheme. If the field is numeric, such as size, the new field will be Anomaly_Score_Num(size). If the field is non-numeric, such as name, the new field will be Anomaly_Score_Cat(name).

Syntax

anomalousvalue <av-options>... [action] [pthresh] [field-list]

Required arguments

None.

Optional arguments

<av-options>

Syntax: minsupcount=<int> | maxanofreq=<float> | minsupfreq=<float> | minnormfreq=<float>

Description: Specify one or more option to control which fields are considered for discriminating anomalies.

Descriptions for the av-option arguments

See also

anomalousvalue, cluster, kmeans, outlier

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the anomalies command.
maxanofreq

Syntax: maxanofreq=<float>
Description: Maximum anomalous frequency is expressed as a floating point number between 0 and 1. Omits a field from consideration if the field is too frequently anomalous. If the ratio of anomalous occurrences of the field to the total number of occurrences of the field is greater than the maxanofreq value, then the field is removed from consideration.
Default: 0.05

minnormfreq

Syntax: minnormfreq=<float>
Description: Minimum normal frequency is expressed as a floating point number between 0 and 1. Omits a field from consideration if the field is not anomalous frequently enough. If the ratio of anomalous occurrences of the field to the total number of occurrences of the field is smaller than p, then the field is removed from consideration.
Default: 0.01

minsupcount

Syntax: minsupcount=<int>
Description: Minimum supported count must be a positive integer. Drops a field that has a small number of occurrences in the input result set. If the field appears fewer than N times in the input events, the field is removed from consideration.
Default: 100

minsupfreq

Syntax: minsupfreq=<float>
Description: Minimum supported frequency is expressed as a floating point number between 0 and 1. Drops a field that has a low frequency of occurrence. The minsupfreq argument checks the ratio of occurrences of the field to the total number of events. If this ratio is smaller than p the field is removed from consideration.
Default: 0.05

action

Syntax: action=annotate | filter | summary
Description: Specify whether to return the anomaly score (annotate), filter out events that are not anomalous values (filter), or return a summary of anomaly statistics (summary).
Default: filter
Descriptions for the action arguments

annotate

**Syntax:** action=annotate

**Description:** The `annotate` action adds new fields to the events containing anomalous values. The fields that are added are `Anomaly_Score_Cat(field)`, `Anomaly_Score_Num(field)`, or both.

filter

**Syntax:** action=filter

**Description:** The `filter` action returns events with anomalous values. Events without anomalous values are removed. The events that are returned are annotated, as described for `action=annotate`.

summary

**Syntax:** action=summary

**Description:** The `summary` action returns a table summarizing the anomaly statistics for each field generated. The table includes how many events contained this field, the fraction of events that were anomalous, what type of test (categorical or numerical) were performed, and so on.

<table>
<thead>
<tr>
<th>Output field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldname</td>
<td>The name of the field.</td>
</tr>
<tr>
<td>count</td>
<td>The number of times the field appears.</td>
</tr>
<tr>
<td>distinct_count</td>
<td>The number of unique values of the field.</td>
</tr>
<tr>
<td>mean</td>
<td>The calculated mean of the field values.</td>
</tr>
<tr>
<td>catAnoFreq%</td>
<td>The anomalous frequency of the categorical field.</td>
</tr>
<tr>
<td>catNormFreq%</td>
<td>The normal frequency of the categorical field.</td>
</tr>
<tr>
<td>numAnoFreq%</td>
<td>The anomalous frequency of the numerical field.</td>
</tr>
<tr>
<td>stdev</td>
<td>The standard deviation of the field value.</td>
</tr>
<tr>
<td>supportFreq%</td>
<td>The support frequency of the field.</td>
</tr>
<tr>
<td>useCat</td>
<td>Use categorical anomaly detection. Categorical anomaly detection looks for rare values.</td>
</tr>
<tr>
<td>useNum</td>
<td>Use numerical anomaly detection. Numerical anomaly detection looks for values that are far from the mean value. This anomaly detection is</td>
</tr>
<tr>
<td>Output field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Gaussian distribution based.</td>
<td></td>
</tr>
<tr>
<td>isNum</td>
<td>Whether or not the field is numerical.</td>
</tr>
</tbody>
</table>

**field-list**

**Syntax:** `<field> ...`

**Description:** The List of fields to consider.

**Default:** If no field list is provided, all fields are considered.

**pthresh**

**Syntax:** `pthresh=<num>`

**Description:** Probability threshold (as a decimal) that has to be met for a value to be considered anomalous.

**Default:** 0.01.

**Usage**

By default, a maximum of 50,000 results are returned. This maximum is controlled by the `maxresultrows` setting in the `[anomalousvalue]` stanza in the `limits.conf` file. Increasing this limit can result in more memory usage.

Only users with file system access, such as system administrators, can edit the configuration files. Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

See How to edit a configuration file.

**Basic examples**

1. **Return only uncommon values from the search results**

   ```
   ... | anomalousvalue
   ```

   This is the same as running the following search:

   ```
   ...| anomalousvalue action=filter pthresh=0.01
   ```

2. **Return uncommon values from the host "reports"**

   ```
   host="reports" | anomalousvalue action=filter pthresh=0.02
   ```
Extended example

1. **Return a summary of the anomaly statistics for each numeric field**

This search uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the USGS Earthquake Feeds and upload the file to your Splunk instance. This example uses the All Earthquakes data from the past 30 days.

Search for anomalous values in the earthquake data.

```
source="all_month.csv"| anomalousvalue action=summary pthresh=0.02 | search isNum=YES
```

The numeric results are returned with multiple decimals. Use the field formatting icon, which looks like a pencil, to enable number formatting and specify the decimal precision to display.
See also

analyzefields, anomalies, cluster, kmeans, outlier

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the anomalousvalue command.

anomalydetection

Description

A streaming and reporting command that identifies anomalous events by computing a probability for each event and then detecting unusually small probabilities. The probability is defined as the product of the frequencies of each individual field value in the event.

- For categorical fields, the frequency of a value X is the number of times X occurs divided by the total number of events.
- For numerical fields, we first build a histogram for all the values, then compute the frequency of a value X as the size of the bin that contains X divided by the number of events.

The anomalydetection command includes the capabilities of the existing anomalousvalue and outlier commands and offers a histogram-based approach for detecting anomalies.
Syntax

anomalydetection [<method-option>] [<action-option>] [<pthresh-option>] [<cutoff-option>] [<field-list>]

Optional arguments

<method-option>
  Syntax: method = histogram | zscore | iqr
  Description: Select the method of anomaly detection. When method=zscore, performs like the anomalousvalue command. When method=iqr, performs like the outlier command. See Usage.
  Default: method=histogram

<action-option>
  Syntax for method=histogram or method=zscore: action = filter | annotate | summary
  Syntax for method=iqr: action = remove | transform
  Description: The actions and defaults depend on the method that you specify. See the detailed descriptions for the actions for each method below.

<pthresh-option>
  Syntax: pthresh=<num>
  Description: Used with method=histogram or method=zscore. Sets the probability threshold, as a decimal number, that has to be met for an event to be deemed anomalous.
  Default: For method=histogram, the command calculates pthresh for each data set during analysis. For method=zscore, the default is 0.01. If you try to use this when method=iqr, it returns an invalid argument error.

<cutoff-option>
  Syntax: cutoff=<bool>
  Description: Sets the upper bound threshold on the number of anomalies. This option applies to only the histogram method. If cutoff=false, the algorithm uses the formula threshold = 1st-quartile - 1.5 * IRQ without modification. If cutoff=true, the algorithm modifies the formula in order to come up with a smaller number of anomalies.
  Default: true

<field-list>
  Syntax: <string> <string> ...
  Description: A list of field names.
Histogram actions

[action-option>

Syntax: action=annotate | filter | summary

Description: Specifies whether to return all events with additional fields (annotate), to filter out events with anomalous values (filter), or to return a summary of anomaly statistics (summary).

Default: filter

When action=filter, the command returns anomalous events and filters out other events. Each returned event contains four new fields. When action=annotate, the command returns all the original events with the same four new fields added when action=filter.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_event_prob</td>
<td>The natural logarithm of the event probability.</td>
</tr>
<tr>
<td>probable_cause</td>
<td>The name of the field that best explains why the event is anomalous. No one field causes anomaly by itself, but often some field value occurs too rarely to make the event probability small.</td>
</tr>
<tr>
<td>probable_cause_freq</td>
<td>The frequency of the value in the probable_cause field.</td>
</tr>
<tr>
<td>max_freq</td>
<td>Maximum frequency for all field values in the event.</td>
</tr>
</tbody>
</table>

When action=summary, the command returns a single event containing six fields.

<table>
<thead>
<tr>
<th>Output field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_anomalies</td>
<td>The number of anomalous events.</td>
</tr>
<tr>
<td>thresh</td>
<td>The event probability threshold that separates anomalous events.</td>
</tr>
<tr>
<td>max_logprob</td>
<td>The maximum of all $\log(\text{event_prob})$.</td>
</tr>
<tr>
<td>min_logprob</td>
<td>The minimum of all $\log(\text{event_prob})$.</td>
</tr>
<tr>
<td>1st_quartile</td>
<td>The first quartile of all $\log(\text{event_prob})$.</td>
</tr>
<tr>
<td>3rd_quartile</td>
<td>The third quartile of all $\log(\text{event_prob})$.</td>
</tr>
</tbody>
</table>

Zscore actions

[action-option>

Syntax: action=annotate | filter | summary

Description: Specifies whether to return the anomaly score (annotate), filter out events with anomalous values (filter), or a summary of anomaly
statistics (summary).

Default: filter

When action=filter, the command returns events with anomalous values while other events are dropped. The kept events are annotated, like the annotate action.

When action=annotate, the command adds new fields, Anomaly_Score_Cat(field) and Anomaly_Score_Num(field), to the events that contain anomalous values.

When action=summary, the command returns a table that summarizes the anomaly statistics for each field is generated. The table includes how many events contained this field, the fraction of events that were anomalous, what type of test (categorical or numerical) were performed, and so on.

**IQR actions**

<action-option>

Syntax: action=remove | transform

Description: Specifies what to do with outliers. The remove action removes the event containing the outlying numerical value. The transform action transforms the event by truncating the outlying value to the threshold for outliers. If mark=true, the transform action prefixes the value with "000".

Abbreviations: The abbreviation for remove is rm. The abbreviation for transform is tf.

Default: action=transform

**Usage**

The anomalydetection command is a **streaming command** command. See Command types.

**The zscore method**

When you specify method=zscore, the anomalydetection command performs like the anomalousvalue command. You can specify the syntax components of the anomalousvalue command when you use the anomalydetection command with method=zscore. See the anomalousvalue command.
**The iqr method**

When you specify `method=iqr`, the `anomalydetection` command performs like the `outlier` command. You can specify the syntax components of the `outlier` command when you specify `method=iqr` with the `anomalydetection` command. For example, you can specify the outlier options `<action>`, `<mark>`, `<param>`, and `<uselower>`. See the `outlier` command.

**Examples**

**Example 1: Return only anomalous events**

These two searches return the same results. The arguments specified in the second search are the default values.

```
... | anomalydetection
```

```
... | anomalydetection method=histogram action=filter
```

**Example 2: Return a short summary of how many anomalous events are there**

Return a short summary of how many anomalous events are there and some other statistics such as the threshold value used to detect them.

```
... | anomalydetection action=summary
```

**Example 3: Return events with anomalous values**

This example specifies `method=zscore` to return anomalous values. The search uses the `filter` action to filter out events that do not have anomalous values. Events must meet the probability threshold `pthresh` before being considered an anomalous value.

```
... | anomalydetection method=zscore action=filter pthresh=0.05
```

**Example 4: Return outliers**

This example uses the outlier options from the `outlier` command. The abbreviation `tf` is used for the transform action in this example.

```
... | anomalydetection method=iqr action=tf param=4 uselower=true mark=true
```
See also

analyzefields, anomalies, anomalousvalue, cluster, kmeans, outlier

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the anomalydetection command.

append

Description

Appends the results of a subsearch to the current results. The append command runs only over historical data and does not produce correct results if used in a real-time search.

For more information about when to use the append command, see the flowchart in the topic About event grouping and correlation in the Search Manual.

If you are familiar with SQL but new to SPL, see Splunk SPL for SQL users.

Syntax

append [...] <subsearch>

Required arguments

subsearch

Syntax: [subsearch]
Description: A secondary search where you specify the source of the events that you want to append. The subsearch must be enclosed in square brackets. See About subsearches in the Search Manual.

Optional arguments

subsearch-options

Syntax: extenderange=<boolean> | maxtime=<int> | maxout=<int> | timeout=<int>
Description: Controls how the subsearch is processed.
**Subsearch options**

extendtimerange

Syntax: extendtimerange=<boolean>

Description: Specifies whether to include the subsearch time range in the time range for the entire search. Use the `extendtimerange` argument when the time range in the subsearch extends beyond the time range for the main search. Use this argument when a **transforming command**, such as `chart`, `timechart`, or `stats`, follows the `append` command in the search and the search uses time based bins.

Default: false

maxtime

Syntax: maxtime=<int>

Description: The maximum time, in seconds, to spend on the subsearch before automatically finalizing.

Default: 60

maxout

Syntax: maxout=<int>

Description: The maximum number of result rows to output from the subsearch.

Default: 50000

timeout

Syntax: timeout=<int>

Description: The maximum time, in seconds, to wait for subsearch to fully finish.

Default: 60

**Usage**

The `append` command is a **streaming command** command. See Command types.

**Examples**

1: Use the `append` command to add column totals.

This search uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., etc.,
for each earthquake recorded.

You can download a current CSV file from the **USGS Earthquake Feeds** and upload the file to your Splunk instance. This example uses the **All Earthquakes** data from the past 30 days.

Count the number of earthquakes that occurred in and around California yesterday and then calculate the total number of earthquakes.

```
source=usgs place=*California* | stats count by magType | append [search index=usgs_* source=usgs place=*California* | stats count]
```

This example uses a subsearch to count all the earthquakes in the California regions (`place=*California*`), then uses the main search to count the number of earthquakes based on the magnitude type of the search.

You cannot use the `stats` command to simultaneously count the total number of events and the number of events for a specified field. The subsearch is used to count the total number of earthquakes that occurred. This count is added to the results of the previous search with the `append` command.

Because both searches share the `count` field, the results of the subsearch are listed as the last row in the count column.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>magType</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>123</td>
</tr>
<tr>
<td>MbLg</td>
<td>1</td>
</tr>
<tr>
<td>Md</td>
<td>1565</td>
</tr>
<tr>
<td>Me</td>
<td>2</td>
</tr>
<tr>
<td>Ml</td>
<td>1202</td>
</tr>
<tr>
<td>Mw</td>
<td>6</td>
</tr>
<tr>
<td>ml</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2909</td>
</tr>
</tbody>
</table>

This search demonstrates how to use the `append` command in a way that is similar to using the `addcoltotals` command to add the column totals.
2. Count the number of different customers who purchased items. Append the top purchaser for each type of product.

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

Count the number of different customers who purchased something from the Buttercup Games online store yesterday, and break this count down by the type of product (accessories, t-shirts, and type of games) they purchased. Also, list the top purchaser for each type of product and how much that person bought of that product.

```
sourcetype=access_* action=purchase | stats dc(clientip) BY categoryId | append [search sourcetype=access_* action=purchase | top 1 clientip BY categoryId] | table categoryId, dc(clientip), clientip, count
```

This example first searches for purchase events (action=purchase). These results are piped into the stats command and the dc(), or distinct_count() function is used to count the number of different users who make purchases. The BY clause is used to break up this number based on the different category of products (categoryId).

This example contains a subsearch as an argument for the append command.

```
...[search sourcetype=access_* action=purchase | top 1 clientip BY categoryId]
```

The subsearch is used to search for purchase events and count the top purchaser (based on clientip) for each category of products. These results are added to the results of the previous search using the append command.

Here, the table command is used to display only the category of products (categoryId), the distinct count of users who bought each type of product (dc(clientip)), the actual user who bought the most of a product type (clientip), and the number of each product that user bought (count).
You can see that the `append` command just tacks on the results of the subsearch to the end of the previous search, even though the results share the same field values. It does not let you manipulate or reformat the output.

### 3. Use the append command to determine the number of unique IP addresses that accessed the Web server.

Use the `append` command, along with the `stats`, `count`, and `top` commands to determine the number of unique IP addresses that accessed the Web server. Find the user who accessed the Web server the most for each type of page request.

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range `Yesterday` when you run the search.

Count the number of different IP addresses that accessed the Web server and also find the user who accessed the Web server the most for each type of page request (`method`).

```
sourcetype=access_* | stats dc(clientip), count by method | append [search sourcetype=access_* | top 1 clientip by method]
```

The Web access events are piped into the `stats` command and the `dc()` or `distinct_count()` function is used to count the number of different users who accessed the site. The `count()` function is used to count the total number of times the site was accessed. These numbers are separated by the page request (`method`).

<table>
<thead>
<tr>
<th>Method</th>
<th>Access Count</th>
<th>User Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESSORIES</td>
<td>37</td>
<td>91.208.94.24</td>
</tr>
<tr>
<td>ARCADE</td>
<td>58</td>
<td>211.166.11.101</td>
</tr>
<tr>
<td>NULL</td>
<td>8</td>
<td>87994.296.51</td>
</tr>
<tr>
<td>SHOOTER</td>
<td>34</td>
<td>87994.296.51</td>
</tr>
<tr>
<td>SIMULATION</td>
<td>34</td>
<td>211.166.11.101</td>
</tr>
<tr>
<td>SPORTS</td>
<td>12</td>
<td>95.163.78.227</td>
</tr>
<tr>
<td>STRATEGY</td>
<td>34</td>
<td>76.165.176.102</td>
</tr>
<tr>
<td>TEE</td>
<td>58</td>
<td>87994.296.51</td>
</tr>
</tbody>
</table>
The subsearch is used to find the top user for each type of page request (method). The append command is used to add the result of the subsearch to the bottom of the table.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>method</th>
<th>dc(clientip)</th>
<th>count</th>
<th>clientip</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>173</td>
<td>2666</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>168</td>
<td>1727</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>83</td>
<td>87.194.216.51</td>
<td>3.113278</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>64</td>
<td>87.194.216.51</td>
<td>3.705848</td>
<td></td>
</tr>
</tbody>
</table>

The first two rows are the results of the first search. The last two rows are the results of the subsearch. Both result sets share the method and count fields.

4. Specify the maximum time for the subsearch to run and the maximum number of result rows from the subsearch

Use the append command, to determine the number of unique IP addresses that accessed the Web server. Find the user who accessed the Web server the most for each type of page request.

This example uses the sample dataset from the Search Tutorial but should work with any format of Apache web access log. Download the data set from this topic in the Search Tutorial and follow the instructions to upload it to your Splunk deployment. Use the time range Yesterday when you run this search.

Count the number of different IP addresses that accessed the Web server and also find the user who accessed the Web server the most for each type of page request (method). Limit the subsearch to 30 seconds and the maximum number of subsearch results to 1000.

sourcetype=access_* | stats dc(clientip), count by method | append maxtime=30 maxout=1000 [search sourcetype=access_* | top 1 clientip by method]

5. Use the extendtimerange argument

Use the extendtimerange argument to ensure that the time range used for the search includes both the time range of the main search and the time range of the subsearch.
The time range used for the search is from 11/1/2017:00:00:00, the earliest time in the subsearch, to 11/30/2017:00:00:00, the latest time in the main search.

See also

appendcols, appendpipe, join, set

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the append command.

appendcols

Description

Appends the fields of the subsearch results with the input search results. External fields of the subsearch that do not start with an underscore character ( _ ) are not combined into the current results. The first subsearch result is merged with the first main result, the second subsearch result is merged with the second main result, and so on.

Syntax

appendcols [override= <bool> | <subsearch-options>...] <subsearch>

Required arguments

subsearch
  Description: A secondary search added to the main search. See how subsearches work in the Search Manual.

Optional arguments

override
  Syntax: override=<bool>
**Description:** If the `override` argument is false, and if a field is present in both a subsearch result and the main result, the main result is used. If `override=true`, the subsearch result value is used.

**Default:** `override=false`

**subsearch-options**

**Syntax:** `maxtime=<int> | maxout=<int> | timeout=<int>`

**Description:** These options control how the subsearch is executed.

**Subsearch options**

**maxtime**

**Syntax:** `maxtime=<int>`

**Description:** The maximum time, in units of seconds, to spend on the subsearch before automatically finalizing.

**Default:** 60

**maxout**

**Syntax:** `maxout=<int>`

**Description:** The maximum number of result rows to output from the subsearch.

**Default:** 50000

**timeout**

**Syntax:** `timeout=<int>`

**Description:** The maximum time, in units of seconds, to wait for subsearch to fully finish.

**Default:** 60

**Examples**

**Example 1:**

Search for "404" events and append the fields in each event to the previous search results.

```
... | appendcols [search 404]
```

**Example 2:**

This search uses `appendcols` to count the number of times a certain field occurs on a specific server and uses that value to calculate other fields.
specific.server | stats dc(userID) as totalUsers | appendcols [ search specific.server AND "text" | addinfo | where _time >= info_min_time AND _time <= info_max_time | stats count(field) as variableA ] | eval variableB = exact(variableA/totalUsers)

- First, this search uses `stats` to count the number of individual users on a specific server and names that variable "totalUsers".
- Then, this search uses `appendcols` to search the server and count how many times a certain field occurs on that specific server. This count is renamed "VariableA". The `addinfo` command is used to constrain this subsearch within the range of `info_min_time` and `info_max_time`.
- The `eval` command is used to define a "variableB".

The result is a table with the fields totalUsers, variableA, and variableB.

See also
append, appendpipe, join, set

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the appendcols command.

appendpipe

Description

Appends the result of the subpipeline to the search results. Unlike a subsearch, the subpipeline is not run first. The subpipeline is run when the search reaches the `appendpipe` command. The `appendpipe` command is used to append the output of transforming commands, such as chart, timechart, stats, and top.

Syntax

appendpipe [run_in_preview=<bool>] [<subpipeline>]
Optional Arguments

run_in_preview

**Syntax:** run_in_preview=<bool>

**Description:** Specifies whether or not display the impact of the `appendpipe` command in the preview. When set to FALSE, the search runs and the preview shows the results as if the `appendpipe` command is not part of the search. However, when the search finishes, the results include the impact of the `appendpipe` command.

**Default:** True

subpipeline

**Syntax:** <subpipeline>

**Description:** A list of commands that are applied to the search results from the commands that occur in the search before the `appendpipe` command.

Usage

The `appendpipe` command can be useful because it provides a summary, total, or otherwise descriptive row of the entire dataset when you are constructing a table or chart. This command is also useful when you need the original results for additional calculations.

Examples

**Example 1:**

Append subtotals for each action across all users.

```plaintext
index=_audit | stats count by action user | appendpipe [stats sum(count) as count by action | eval user = "TOTAL - ALL USERS"] | sort action
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>action</th>
<th>user</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>accelerate_search</td>
<td>admin</td>
<td>209</td>
</tr>
<tr>
<td>accelerate_search</td>
<td>buttercup</td>
<td>345</td>
</tr>
<tr>
<td>accelerate_search</td>
<td>can-delete</td>
<td>6</td>
</tr>
<tr>
<td>accelerate_search</td>
<td>TOTAL - ALL USERS</td>
<td>560</td>
</tr>
<tr>
<td>action</td>
<td>user</td>
<td>count</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>add</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td>add</td>
<td>TOTAL - ALL USERS</td>
<td>1</td>
</tr>
<tr>
<td>change_authentication</td>
<td>admin</td>
<td>50</td>
</tr>
<tr>
<td>change_authentication</td>
<td>buttercup</td>
<td>9</td>
</tr>
<tr>
<td>change_authentication</td>
<td>can-delete</td>
<td>24</td>
</tr>
<tr>
<td>change_authentication</td>
<td>TOTAL - ALL USERS</td>
<td>83</td>
</tr>
</tbody>
</table>

**See also**

append, appendcols, join, set

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the appendpipe command.

**arules**

**Description**

The arules command looks for associative relationships between field values. The command returns a table with the following columns: Given fields, Implied fields, Strength, Given fields support, and Implied fields support. The given and implied field values are the values of the fields you supply. The Strength value indicates the relationship between (among) the given and implied field values.

Implements arules algorithm as discussed in Michael Hahsler, Bettina Gruen and Kurt Hornik (2012). arules: Mining Association Rules and Frequent Itemsets. R package version 1.0-12. This algorithm is similar to the algorithms used for online shopping websites which suggest related items based on what items other customers have viewed or purchased.

**Syntax**

arules [<arules-option>...] <field-list>...
**Required arguments**

field-list

**Syntax:** <field> <field> ...

**Description:** The list of field names. At least two fields must be specified.

**Optional arguments**

<arules-option>

**Syntax:** <support> | <confidence>

**Description:** Options for arules command.

**arules options**

support

**Syntax:** sup=<int>

**Description:** Specify a support limit. Associations with computed support levels smaller than this value are not included in the output results. The support option must be a positive integer.

**Default:** 3

confidence

**Syntax:** conf=<float>

**Description:** Specify a confidence limit. Associations with a confidence (expressed as Strength field) are not included in the output results. Must be between 0 and 1.

**Default:** .5

**Usage**

The `arules` command is a **streaming command**. See [Command types](#).

**Examples**

**Example 1:** Search for the likelihood that the fields are related.

```
... | arules field1 field2 field3
```

**Example 2:**

```
... | arules sup=3 conf=.6 field1 field2 field3
```

221
See also
associate, correlate

Answers
Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the arules command.

associate

Description
The associate command identifies correlations between fields. The command tries to find a relationship between pairs of fields by calculating a change in entropy based on their values. This entropy represents whether knowing the value of one field helps to predict the value of another field.

In Information Theory, entropy is defined as a measure of the uncertainty associated with a random variable. In this case if a field has only one unique value, the field has an entropy of zero. If the field has multiple values, the more evenly those values are distributed, the higher the entropy.

The associate command uses Shannon entropy (log base 2). The unit is in bits.

Syntax
associate [<associate-options>...] [field-list]

Required arguments
None.

Optional arguments
associate-option
Syntax: supcnt | supfreq | improv
Description: Options for the associate command. See the Associate-options section.

field-list
Syntax: <field> ...

Description: A list of one or more fields. You cannot use wildcard characters in the field list. If you specify a list of fields, the analysis is restricted to only those fields.
Default: All fields are analyzed.

Associate-options

supcnt

Syntax: supcnt=<num>

Description: Specifies the minimum number of times that the "reference key=reference value" combination must appear. Must be a non-negative integer.
Default: 100

supfreq

Syntax: supfreq=<num>

Description: Specifies the minimum frequency of "reference key=reference value" combination as a fraction of the number of total events.
Default: 0.1

improv

Syntax: improv=<num>

Description: Specifies a limit, or minimum entropy improvement, for the "target key". The calculated entropy improvement must be greater than or equal to this limit.
Default: 0.5

Columns in the output table

The associate command outputs a table with columns containing the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference_Key</td>
<td>The <em>name</em> of the first field in a pair of fields.</td>
</tr>
<tr>
<td>Reference_Value</td>
<td>The <em>value</em> in the first field in a pair of fields.</td>
</tr>
<tr>
<td>Target_Key</td>
<td>The name of the second field in a pair of fields.</td>
</tr>
<tr>
<td>Unconditional_Entropy</td>
<td>The entropy of the target key.</td>
</tr>
<tr>
<td>Conditional_Entropy</td>
<td>The entropy of the target key when the reference key is the reference value.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Entropy_Improvement</td>
<td>The difference between the unconditional entropy and the conditional entropy.</td>
</tr>
<tr>
<td>Description</td>
<td>A message that summarizes the relationship between the field values that is based on the entropy calculations. The Description is a textual representation of the result. It is written in the format: &quot;When the 'Reference_Key' has the value 'Reference_Value', the entropy of 'Target_Key' decreases from Unconditional_Entropy to Conditional_Entropy.&quot;</td>
</tr>
<tr>
<td>Support</td>
<td>Specifies how often the reference field is the reference value, relative to the total number of events. For example, how often field A is equal to value X, in the total number of events.</td>
</tr>
</tbody>
</table>

**Examples**

1. **Analyze the relationship between fields in web access log files**

   This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range **Yesterday** when you run the search.

   This example demonstrates one way to analyze the relationship of fields in your web access logs.

   ```splunk
   sourcetype=access_* status!=200 | fields method, status | associate improv=0.05 | table Reference_Key, Reference_Value, Target_Key, Top_Conditional_Value, Description
   ```

   The first part of this search retrieves web access events that returned a status that is not 200. Web access data contains many fields. You can use the associate command to see a relationship between all pairs of fields and values in your data. To simplify this example, restrict the search to two fields: method and status.

   Because the associate command adds many columns to the output, this search uses the table command to display only select columns.

   The results appear on the Statistics tab and look something like this:

   ![Results](image-url)
<table>
<thead>
<tr>
<th>Reference_Key</th>
<th>Reference_Value</th>
<th>Target_Key</th>
<th>Top_Conditional_Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>POST</td>
<td>status</td>
<td>503 (17.44% -&gt; 33.96%)</td>
<td>When 'method' has the value 'POST', the entropy of 'status' decreases from 2.923 to 2.729.</td>
</tr>
<tr>
<td>status</td>
<td>400</td>
<td>method</td>
<td>GET (76.37% -&gt; 83.45%)</td>
<td>When 'status' has the value '400', the entropy of 'method' decreases from 0.789 to 0.647.</td>
</tr>
<tr>
<td>status</td>
<td>404</td>
<td>method</td>
<td>GET (76.37% -&gt; 81.27%)</td>
<td>When 'status' has the value '404', the entropy of 'method' decreases from 0.789 to 0.696.</td>
</tr>
<tr>
<td>status</td>
<td>406</td>
<td>method</td>
<td>GET (76.37% -&gt; 81.69%)</td>
<td>When 'status' has the value '406', the entropy of 'method' decreases from 0.789 to 0.687.</td>
</tr>
<tr>
<td>status</td>
<td>408</td>
<td>method</td>
<td>GET (76.37% -&gt; 80.00%)</td>
<td>When 'status' has the value</td>
</tr>
</tbody>
</table>
In the results you can see that there is one method and five status values in the results.

From the first row of results, you can see that when method=POST, the status field is 503 for those events. The associate command concludes that, if method=POST, the Top_Conditional_Value is likely to be 503 as much as 33% of the time.

The Reference_Key and Reference_Value are being correlated to the Target_Key.

The Top_Conditional_Value field states three things:

- The most common value for the given Reference_Value
- The frequency of the Reference_Value for that field in the dataset
- The frequency of the most common associated value in the Target_Key for the events that have the specific Reference_Value in that Reference Key.

It is formatted to read "CV (FRV% -> FCV%)" where CV is the conditional Value, FRV is is the percentage occurrence of the reference value, and FCV is the percentage of occurrence for that conditional value, in the case of the reference value.
2. **Return results that have at least 3 references to each other**

Return results associated with each other (that have at least 3 references to each other).

```
index=_internal sourcetype=splunkd | associate supcnt=3
```

3. **Analyze events from a host**

Analyze all events from host "reports" and return results associated with each other.

```
host="reports" | associate supcnt=50 supfreq=0.2 improv=0.5
```

**See also**

arules, correlate, contingency

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the associate command.

**audit**

**Description**

Returns audit trail information that is stored in the local `audit` index. This command also validates signed audit events while checking for gaps and tampering.

**Syntax**

audit

**Examples**

**Example 1:** View information in the "audit" index.

```
index="_audit" | audit
```
Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the audit command.

autoregress

Description

Prepares your events for calculating the autoregression, or the moving average, by copying one or more of the previous values for field into each event.

The first few events will lack the augmentation of prior values, since the prior values do not exist.

Syntax

autoregress <field> [AS <newfield>] [ p=<int> | p=<int>-<int> ]

Required arguments

field

Syntax: <string>
Description: The name of a field. Most usefully a field with numeric values.

Optional arguments

p

Syntax: p=<int> | p=<int>-<int>
Description: Specifies which prior events to copy values from. You can specify a single integer or a numeric range. For a single value, such as 3, the autoregress command copies field values from the third prior event into a new field. For a range, the autoregress command copies field values from the range of prior events. For example, if you specify a range such as p=2-4, then the field values from the second, third, and fourth prior events are copied into new fields.
Default: 1

newfield

Syntax: <field>
Description: If \( p \) is set to a single integer, the newfield argument specifies a field name to copy the single field value into. Invalid if \( p \) is set to a range.

If the newfield argument is not specified, the single or multiple values are copied into fields with the names \(<field>_p<num>\). For example, if \( p=2-4 \) and field=count, the field names are count_p2, count_p3, count_p4.

Usage

The autoregress command is a centralized streaming command. See Command types.

Examples

Example 1:

For each event, copy the 3rd previous value of the 'ip' field into the field 'old_ip'.

... | autoregress ip AS old_ip p=3

Example 2:

For each event, copy the 2nd, 3rd, 4th, and 5th previous values of the 'count' field.

... | autoregress count p=2-5

Since the new field argument is not specified, the values are copied into the fields 'count_p2', 'count_p3', 'count_p4', and 'count_p5'.

Example 3:

Calculate a moving average of event size over the current event and the four prior events. This search omits the moving_average for the initial events, where the field would be wrong, because summing null fields is considered null.

... | eval rawlen=len(_raw) | autoregress rawlen p=1-4 | eval moving_average=(rawlen + rawlen_p1 + rawlen_p2 + rawlen_p3 + rawlen_p4) / 5
See also

accum, delta, streamstats, trendline

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the autoregress command.

awssnsalert

The awssnsalert command is used with the Splunk Add-on for AWS.

For information about this command, see Use the awssnsalert search command in Splunk Add-on for AWS.

bin

The bucket command is an alias for the bin command.

Description

Puts continuous numerical values into discrete sets, or bins, by adjusting the value of <field> so that all of the items in a particular set have the same value.

The bin command is automatically called by the chart and the timechart commands. Use the bin command for only statistical operations that the chart and the timechart commands cannot process. Do not use the bin command if you plan to export all events to CSV or JSON file formats.

Syntax

bin [ <bin-options>...] <field> [AS <newfield>]

Required arguments

field

Syntax: <field>
Description: Specify a field name.
Optional arguments

bin-options

Syntax: bins | minspan | span | <start-end> | aligntime
Description: Discretization options. See the Bins options section in this topic for the syntax and description for each of these options.

newfield

Syntax: <string>
Description: A new name for the field.

Bin options

bins

Syntax: bins=<int>
Description: Sets the maximum number of bins to discretize into.

minspan

Syntax: minspan=<span-length>
Description: Specifies the smallest span granularity to use automatically inferring span from the data time range.

span

Syntax: span = <log-span> | <span-length>
Description: Sets the size of each bin, using a span length based on time or logarithm-based span.

<start-end>

Syntax: start=<num> | end=<num>
Description: Sets the minimum and maximum extents for numerical bins. The data in the field is analyzed and the beginning and ending values are determined. The start and end arguments are used when a span value is not specified.

You can use the start or end arguments only to expand the range, not to shorten the range. For example, if the field represents seconds the values are from 0-59. If you specify a span of 10, then the bins are calculated in increments of 10. The bins are 0-9, 10-19, 20-29, and so forth. If you do not specify a span, but specify end=1000, the bins are calculated based on the actual beginning value and 1000 as the end value.

If you set end=10 and the values are >10, the end argument has no effect.
aligntime

Syntax: aligntime=(earliest | latest | <time-specifier>)

Description: Align the bin times to something other than base UTC time (epoch 0). The aligntime option is valid only when doing a time-based discretization. Ignored if span is in days, months, or years.

Span options

log-span

Syntax: [<num>]log[<num>]

Description: Sets to log-based span. The first number is a coefficient. The second number is the base. If the first number is supplied, it must be a real number >= 1.0 and < base. Base, if supplied, must be real number > 1.0 (strictly greater than 1).

Example: span=2log10

span-length

Syntax: <int>[<timescale>]

Description: A span of each bin. If discretizing based on the _time field or used with a timescale, this is treated as a time range. If not, this is an absolute bin length.

<timescale>

Syntax: <sec> | <min> | <hr> | <day> | <month> | <subseconds>

Description: Time scale units. If discretizing based on the _time field.

Default: sec

<table>
<thead>
<tr>
<th>Time scale</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sec&gt;</td>
<td>s</td>
<td>sec</td>
</tr>
<tr>
<td>&lt;min&gt;</td>
<td>m</td>
<td>min</td>
</tr>
<tr>
<td>&lt;hr&gt;</td>
<td>h</td>
<td>hr</td>
</tr>
<tr>
<td>&lt;day&gt;</td>
<td>d</td>
<td>day</td>
</tr>
<tr>
<td>&lt;month&gt;</td>
<td>mon</td>
<td>month</td>
</tr>
<tr>
<td>Time scale</td>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>&lt;subseconds&gt;</td>
<td>us</td>
<td>ms</td>
</tr>
</tbody>
</table>

**Usage**

The `bin` command is usually a **dataset processing command**. If the `span` argument is specified with the command, the `bin` command is a **streaming command**. See **Command types**.

**Examples**

**Example 1:**

Return the average "thruput" of each "host" for each 5 minute time span.

```plaintext
... | bin _time span=5m | stats avg(thruput) by _time host
```

**Example 2:**

Bin search results into 10 bins, and return the count of raw events for each bin.

```plaintext
... | bin size bins=10 | stats count(_raw) by size
```

**Example 3:**

Create bins with an end value larger than you need, ensure that all possible values are included.

```plaintext
... | bin amount end=1000
```

**Example 4:**

Align the time bins to 3am (local time). Set the span to 12h. The bins will represent 3am - 3pm, then 3pm - 3am (the next day), and so on.

```plaintext
... | bin _time span=12h aligntime=@d+3h
```
Example 5:

Align the bins to the specific UTC time of 1500567890.

...| bin _time aligntime=1500567890

See also

chart, timechart

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the bin command.

bucket

The `bucket` command is an alias for the `bin` command. See the `bin` command for the syntax and examples.

See also

bin, chart, timechart

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the bin command.

bucketdir

Description

Replaces a field value with higher-level grouping, such as replacing filenames with directories.

Returns the maxcount events, by taking the incoming events and rolling up multiple sources into directories, by preferring directories that have many files but few events. The field with the path is PATHFIELD (e.g., source), and strings are
broken up by a separator character. The default pathfield=source; sizefield=totalCount; maxcount=20; countfield=totalCount; sep="/" or "\\", depending on the operation system.

**Syntax**

bucketdir pathfield=<field> sizefield=<field> [maxcount=<int>] [countfield=<field>] [sep=<char>]

**Required arguments**

pathfield

Syntax: pathfield=<field>
Description: Specify a field name that has a path value.

sizefield

Syntax: sizefield=<field>
Description: Specify a numeric field that defines the size of bucket.

**Optional arguments**

countfield

Syntax: countfield=<field>
Description: Specify a numeric field that describes the count of events.

maxcount

Syntax: maxcount=<int>
Description: Specify the total number of events to bucket.

sep

Syntax: <char>
Description: The separating character. Specify either a forward slash "/" or double back slashes "\\", depending on the operating system.

**Usage**

The bucketdir command is a **streaming command**. See [Command types](#).

**Examples**

235
Example 1:

Return 10 best sources and directories.

... | top source | bucketdir pathfield=source sizefield=count maxcount=10

See also

cluster, dedup

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the bucketdir command.

cefout

The cefout command is used with the Splunk App for CEF.

For information about this command, see Technical implementation in Deploy and Use Splunk App for CEF.

chart

Description

The chart command is a transforming command that returns your results in a table format. The results can then be used to display the data as a chart, such as a column, line, area, or pie chart. See the Visualization Reference in the Dashboards and Visualizations manual.

You must specify a statistical function when you use the chart command. See Statistical and charting functions.

Syntax

    chart [chart-options>] [agg=<stats-agg-term>]
    ( <stats-agg-term> | <sparkline-agg-term> | "("<eval-expression>")" )...
Required arguments

You must include one of the following arguments when you use the chart command.

stats-agg-term

Syntax: <stats-func> ( <evaled-field> | <wc-field> ) [AS <wc-field>]
Description: A statistical aggregation function. See Stats function options.
The function can be applied to an eval expression, or to a field or set of fields. Use the AS clause to place the result into a new field with a name that you specify. You can use wild card characters in field names.

sparkline-agg-term

Syntax: <sparkline-agg> [AS <wc-field>]
Description: A sparkline aggregation function. Use the AS clause to place the result into a new field with a name that you specify. You can use wild card characters in field names. See Sparkline options.

eval-expression

Syntax: <eval-math-exp> | <eval-concat-exp> | <eval-compare-exp> | <eval-bool-exp> | <eval-function-call>
Description: A combination of literals, fields, operators, and functions that represent the value of your destination field. For more information, see the Evaluation functions. See Usage.

For these evaluations to work, your values need to be valid for the type of operation. For example, with the exception of addition, arithmetic operations might not produce valid results if the values are not numerical. If both operands are strings, they can be concatenated. When concatenating values with a period, the search treats both values as strings regardless of their actual type.

Optional arguments

agg

Syntax: agg=<stats-agg-term>
Description: Specify an aggregator or function. For a list of stats functions with descriptions and examples, see Statistical and charting functions.
chart-options

**Syntax:** cont | format | limit | sep  
**Description:** Options that you can specify to refine the result. See the Chart options section in this topic.  
**Default:**

column-split

**Syntax:** <field> [tc-options]... [where-clause]  
**Description:** Specifies a field to use as the columns in the result table. By default, when the result are visualized, the columns become the data series in the chart. If the field is numerical, default discretization is applied as defined with the tc-options argument. See the tc options and the where clause sections in this topic.  
**Default:** The number of columns included is limited to 10 by default. You can change the number of columns by including a <where-clause>.

**Note:** When a column-split field is included, the output is a table where each column represents a distinct value of the split-by field. This is in contrast with the by-clause, where each row represents a single unique combination of values of the group-by fields. For additional information see the Usage section in this topic.

dedup_splitvals

**Syntax:** dedup_splitvals=<boolean>  
**Description:** Specifies whether to remove duplicate values in multivalued BY clause fields.  
**Default:** false

row-split

**Syntax:** <field> [bin-options]...  
**Description:** The field that you specify becomes the first column in the results table. The field values become the row labels in the results table. In a chart, the field name is used to label the X-axis. The field values become the X-axis values. See the Bin options section in this topic.  
**Default:** None.

**Chart options**

cont

**Syntax:** cont=<bool>  
**Description:** Specifies if the bins are continuous. If cont=false, replots the x-axis so that a noncontinuous sequence of x-value bins show up adjacently in the output. If cont=true, bins that have no values will display...
with a count of 0 or null values.

**Default:** true

**format**

**Syntax:** format=<string>

**Description:** Used to construct output field names when multiple data series are used in conjunction with a split-by-field. format takes precedence over sep and allows you to specify a parameterized expression with the stats aggregator and function ($AGG$) and the value of the split-by-field ($VAL$).

**limit**

**Syntax:** limit=<int>

**Description:** Only valid when a column-split is specified. Use the limit option to specify the number of results that should appear in the output. When you set limit=N the top N values are retained, based on the sum of each series. If limit=0, all results are returned.

**sep**

**Syntax:** sep=<string>

**Description:** Used to construct output field names when multiple data series are used in conjunctions with a split-by field. This is equivalent to setting format to $AGG$<sep>$VAL$.

### Stats function options

**stats-func**

**Syntax:** The syntax depends on the function you use. Refer to the table below.

**Description:** Statistical and charting functions that you can use with the chart command. Each time you invoke the chart command, you can use one or more functions. However, you can only use one BY clause. See Usage.

The following table lists the supported functions by type of function. Use the links in the table to see descriptions and examples for each function. For an overview about using functions with commands, see Statistical and charting functions.
### Supported functions and syntax

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aggregate functions</strong></td>
<td>avg()</td>
</tr>
<tr>
<td></td>
<td>count()</td>
</tr>
<tr>
<td></td>
<td>distinct_count()</td>
</tr>
<tr>
<td></td>
<td>estdc()</td>
</tr>
<tr>
<td></td>
<td>estdc_error()</td>
</tr>
<tr>
<td><strong>Event order functions</strong></td>
<td>earliest()</td>
</tr>
<tr>
<td><strong>Multivalue stats and chart functions</strong></td>
<td>list(X)</td>
</tr>
</tbody>
</table>

### Sparkline options

Sparklines are inline charts that appear within table cells in search results and display time-based trends associated with the primary key of each row.

**sparkline-agg**

**Syntax:** `sparkline (count(<wc-field>), <span-length>) | sparkline (<sparkline-func>(<wc-field>), <span-length>)`

**Description:** A sparkline specifier, which takes the first argument of an aggregation function on a field and an optional timespan specifier. If no timespan specifier is used, an appropriate timespan is chosen based on the time range of the search. If the sparkline is not scoped to a field, only the count aggregate function is permitted. You can use wild card characters in field names.

**span-length**

See the **Span options** section in this topic.

**sparkline-func**

**Syntax:** `c() | count() | dc() | mean() | avg() | stdev() | stdevp() | var() | varp() | sum() | sumsq() | min() | max() | range()`

**Description:** Aggregation function to use to generate sparkline values. Each sparkline value is produced by applying this aggregation to the events that fall into each particular time bin.
For more information see Add sparklines to your search results in the *Search Manual*.

**Bin options**

**Syntax:** bins | span | <start-end> | aligntime  
**Description:** Discretization options.  
**Default:** bins=300

**bins**

**Syntax:** bins=<int>  
**Description:** Sets the maximum number of bins to discretize into. For example, if bin=300, the search finds the smallest bin size that results in no more than 300 distinct bins.  
**Default:** 300

**span**

**Syntax:** span=<log-span> | span=<span-length>  
**Description:** Sets the size of each bin, using a span length based on time or log-based span. See the *Span options* section in this topic.

**<start-end>**

**Syntax:** end=<num> | start=<num>  
**Description:** Sets the minimum and maximum extents for numerical bins. Data outside of the [start, end] range is discarded.

**aligntime**

**Syntax:** aligntime=(earliest | latest | <time-specifier>)  
**Description:** Align the bin times to something other than base UNIX time (epoch 0). The aligntime option is valid only when doing a time-based discretization. Ignored if span is in days, months, or years.

**Span options**

**<log-span>**

**Syntax:** [<num>]<log>[<num>]  
**Description:** Sets to a logarithm-based span. The first number is a coefficient. The second number is the base. If the first number is supplied, it must be a real number >= 1.0 and < base. Base, if supplied, must be real number > 1.0 (strictly greater than 1).

**span-length**

**Syntax:** <span>[<timescale>]**
Description: A span length based on time.

<span>
Syntax: <int>
Description: The span of each bin. If using a timescale, this is used as a time range. If not, this is an absolute bucket "length."

<timescale>
Syntax: <sec> | <min> | <hr> | <day> | <month> | <subseconds>
Description: Time scale units.

<table>
<thead>
<tr>
<th>Time scale</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sec&gt;</td>
<td>s</td>
<td>sec</td>
</tr>
<tr>
<td>&lt;min&gt;</td>
<td>m</td>
<td>min</td>
</tr>
<tr>
<td>&lt;hr&gt;</td>
<td>h</td>
<td>hr</td>
</tr>
<tr>
<td>&lt;day&gt;</td>
<td>d</td>
<td>day</td>
</tr>
<tr>
<td>&lt;month&gt;</td>
<td>mon</td>
<td>month</td>
</tr>
<tr>
<td>&lt;subseconds&gt;</td>
<td>us</td>
<td>ms</td>
</tr>
</tbody>
</table>

tc options

The tc-options is part of the <column-split> argument.

tc-options
Syntax: <bin-options> | usenull=<bool> | useother=<bool> | nullstr=<string> | otherstr=<string>
Description: Options for controlling the behavior of splitting by a field.

bin-options
See the Bin options section in this topic.

nullstr
Syntax: nullstr=<string>
**Description:** If usenull is true, this series is labeled by the value of the nullstr option, and defaults to NULL.

**otherstr**

**String:** otherstr=<string>

**Description:** If useother is true, this series is labeled by the value of the otherstr option, and defaults to OTHER.

**usenull**

**Syntax:** usenull=<bool>

**Description:** Controls whether or not a series is created for events that do not contain the split-by field.

**useother**

**Syntax:** useother=<bool>

**Description:** Specifies if a series should be added for data series not included in the graph because they did not meet the criteria of the <where-clause>.

**where clause**

The <where-clause> is part of the <column-split> argument.

**where clause**

**Syntax:** <single-agg> <where-comp>

**Description:** Specifies the criteria for including particular data series when a field is given in the tc-by-clause. The most common use of this option is to select for spikes rather than overall mass of distribution in series selection. The default value finds the top ten series by area under the curve. Alternately one could replace sum with max to find the series with the ten highest spikes. This has no relation to the where command.

**single-agg**

**Syntax:** count | <stats-func>(<field>)

**Description:** A single aggregation applied to a single field, including an evaluated field. No wildcards are allowed. The field must be specified, except when using the count aggregate function, which applies to events as a whole.

**<stats-func>**

See the Statistical functions section in this topic.

**<where-comp>**

**Syntax:** <wherein-comp> | <wherethresh-comp>
**Description:** The criteria for the `<where-clause>`.

`<wherein-comp>`

**Syntax:** `(in | notin) (top | bottom)<int>`

**Description:** A grouping criteria for the `<where-clause>`.

The aggregated series value be in or not in some top or bottom grouping.

`<wherethresh-comp>`

**Syntax:** `( < | > ) <num>`

**Description:** A threshold for the `<where-clause>`.

The aggregated series value must be greater than or less than the specified numeric threshold.

**Usage**

The `chart` command is a **transforming command**. See [Command types](#).

**Evaluation expressions**

You can use the `chart` command with an eval expression. Unless you specify a `split-by` clause, the eval expression must be renamed.

**Functions and memory usage**

Some functions are inherently more expensive, from a memory standpoint, than other functions. For example, the `distinct_count` function requires far more memory than the `count` function. The `values` and `list` functions also can consume a lot of memory.

If you are using the `distinct_count` function without a split-by field or with a low-cardinality split-by by field, consider replacing the `distinct_count` function with the the `estdc` function (estimated distinct count). The `estdc` function might result in significantly lower memory usage and run times.

**X-axis**

You can specify which field is tracked on the x-axis of the chart. The x-axis variable is specified with a `by` field and is discretized if necessary. Charted fields are converted to numerical quantities if necessary.

Unlike the `timechart` command which generates a chart with the `_time` field as the x-axis, the `chart` command produces a table with an arbitrary field as the
You can also specify the x-axis field after the `over` keyword, before any `by` and subsequent `split-by` clause. The `limit` and `agg` options allow easier specification of series filtering. The `limit` and `agg` options are ignored if an explicit `where-clause` is provided.

**Using row-split and column-split fields**

When a column-split field is included, the output is a table where each column represents a distinct value of the column-split field. This is in contrast with the `stats` command, where each row represents a single unique combination of values of the group-by fields. The number of columns included is limited to 10 by default. You can change the number of columns by including a `where-clause`.

With the `chart` and `timechart` commands, you cannot specify the same field in a function and as the row-split field.

For example, you **cannot** run this search. The field `A` is specified in the `sum` function and the `row-split` argument.

```
... | chart sum(A) by A span=log2
```

You must specify a different field as in the `row-split` argument.

Alternatively, you can work around this problem by using an `eval` expression. For example:

```
... | eval A1=A | chart sum(A) by A1 span=log2
```

**Basic Examples**

1. **Chart the max(delay) for each value of foo**

   Return max(delay) for each value of foo.

   ```
   ... | chart max(delay) OVER foo
   ```

2. **Chart the max(delay) for each value of foo, split by the value of bar**

   Return max(delay) for each value of foo split by the value of bar.

   ```
   ... | chart max(delay) OVER foo BY bar
   ```
3. **Chart the ratio of the average to the maximum "delay" for each distinct "host" and "user" pair**

Return the ratio of the average (mean) "size" to the maximum "delay" for each distinct "host" and "user" pair.

```plaintext
... | chart eval(avg(size)/max(delay)) AS ratio BY host user
```

4. **Chart the maximum "delay" by "size" and separate "size" into bins**

Return the maximum "delay" by "size", where "size" is broken down into a maximum of 10 equal sized bins.

```plaintext
... | chart max(delay) BY size bins=10
```

5. **Chart the average size for each distinct host**

Return the average (mean) "size" for each distinct "host".

```plaintext
... | chart avg(size) BY host
```

6. **Chart the number of events, grouped by date and hour**

Return the number of events, grouped by date and hour of the day, using span to group per 7 days and 24 hours per half days. The span applies to the field immediately prior to the command.

```plaintext
... | chart count BY date_mday span=3 date_hour span=12
```

7. **Align the chart time bins to local time**

Align the time bins to 5am (local time). Set the span to 12h. The bins will represent 5am - 5pm, then 5pm - 5am (the next day), and so on.

```plaintext
...| chart _time span=12h aligntime=@d+5h
```

8. **In a multivalue BY field, remove duplicate values**

For each unique value of mvfield, chart the average value of field. Deduplicates the values in the mvfield.

```plaintext
...| chart avg(field) BY mvfield dedup_splitval=true
```
Extended Examples

1. Specify `<row-split>` and `<column-split>` values with the `chart` command

This example uses events that list the numeric sales for each product and quarter, for example:

<table>
<thead>
<tr>
<th>products</th>
<th>quarter</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
<td>QTR1</td>
<td>1200</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR1</td>
<td>1400</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR1</td>
<td>1650</td>
</tr>
<tr>
<td>ProductA</td>
<td>QTR2</td>
<td>1425</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR2</td>
<td>1175</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR2</td>
<td>1550</td>
</tr>
<tr>
<td>ProductA</td>
<td>QTR3</td>
<td>1300</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR3</td>
<td>1250</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR3</td>
<td>1375</td>
</tr>
<tr>
<td>ProductA</td>
<td>QTR4</td>
<td>1550</td>
</tr>
<tr>
<td>ProductB</td>
<td>QTR4</td>
<td>1700</td>
</tr>
<tr>
<td>ProductC</td>
<td>QTR4</td>
<td>1625</td>
</tr>
</tbody>
</table>

To summarize the data by product for each quarter, run this search:

```
source="addtotalsData.csv" | chart sum(sales) BY products quarter
```

In this example, there are two fields specified in the BY clause with the `chart` command.

- The `products` field is referred to as the `<row-split>` field. In the chart, this field forms the X-axis.
- The `quarter` field is referred to as the `<column-split>` field. In the chart, this field forms the data series.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>products</th>
<th>QTR1</th>
<th>QTR2</th>
<th>QTR3</th>
<th>QTR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
<td>1200</td>
<td>1425</td>
<td>1300</td>
<td>1550</td>
</tr>
</tbody>
</table>

247
see the addtotals command for an example that adds a total column for each product.

2. Chart the number of different page requests for each Web server

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

Chart the number of different page requests, GET and POST, that occurred for each Web server.

sourcetype=access_* | chart count(eval(method="GET")) AS GET, count(eval(method="POST")) AS POST by host

This example uses eval expressions to specify the different field values for the stats command to count. The first clause uses the count() function to count the Web access events that contain the method field value GET. Then, using the AS keyword, the field that represents these results is renamed GET.

The second clause does the same for POST events. The counts of both types of events are then separated by the web server, using the BY clause with the host field.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>host</th>
<th>GET</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>wwww1</td>
<td>8431</td>
<td>5197</td>
</tr>
<tr>
<td>wwww2</td>
<td>8097</td>
<td>4815</td>
</tr>
<tr>
<td>wwww3</td>
<td>8338</td>
<td>4654</td>
</tr>
</tbody>
</table>

Click the Visualization tab. If necessary, format the results as a column chart. This chart displays the total count of events for each event type, GET or POST, based on the host value.
3. Chart the number of transactions by duration

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

Create a chart to show the number of transactions based on their duration (in seconds).

```
sourcetype=access_* status=200 action=purchase | transaction clientip maxspan=10m | chart count BY duration span=log2
```

This search uses the `transaction` command to define a transaction as events that share the `clientip` field and fit within a ten minute time span. The `transaction` command creates a new field called `duration`, which is the difference between the timestamps for the first and last events in the transaction. (Because `maxspan=10s`, the `duration` value should not be greater than this.)

The transactions are then piped into the `chart` command. The `count()` function is used to count the number of transactions and separate the count by the duration of each transaction. Because the duration is in seconds and you expect there to be many values, the search uses the `span` argument to bucket the duration into bins of log2 (span=log2).

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>duration</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>970</td>
</tr>
<tr>
<td>1-2</td>
<td>593</td>
</tr>
<tr>
<td>2-4</td>
<td>208</td>
</tr>
</tbody>
</table>
Click the Visualization tab. If necessary, format the results as a column chart.

In this data set, most transactions take between 0 and 2 seconds to complete.

4. Chart the average number of events in a transaction, based on transaction duration

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

Create a chart to show the average number of events in a transaction based on the duration of the transaction.

```
sourcetype=access_* status=200 action=purchase | transaction clientip maxspan=30m | chart avg(eventcount) by duration span=log2
```

The transaction command adds two fields to the results duration and eventcount. The eventcount field tracks the number of events in a single transaction.

In this search, the transactions are piped into the chart command. The avg() function is used to calculate the average number of events for each duration.
Because the duration is in seconds and you expect there to be many values, the search uses the \texttt{span} argument to bucket the duration into bins using logarithm with a base of 2.

Use the field format option to enable number formatting.

Click the \textbf{Visualization} tab and change the display to a pie chart.

Each wedge of the pie chart represents a duration for the event transactions. You can hover over a wedge to see the average values.

\textbf{5. Chart customer purchases}

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to \texttt{get the tutorial data into Splunk}. Use the time range \texttt{Yesterday} when you run the search.
Chart how many different people bought something and what they bought at the Buttercup Games online store yesterday.

```
sourcetype=access_* status=200 action=purchase | chart dc(clientip) OVER date_hour BY categoryId usenull=f
```

This search takes the purchase events and pipes it into the `chart` command. The `dc()` or `distinct_count()` function is used to count the number of unique visitors (characterized by the `clientip` field). This number is then charted over each hour of the day and broken out based on the `categoryId` of the purchase. Also, because these are numeric values, the search uses the `usenull=f` argument to exclude fields that don't have a value.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>date_hour</th>
<th>ACCESSORIES</th>
<th>ARCADE</th>
<th>SHOOTER</th>
<th>SIMULATION</th>
<th>SPORTS</th>
<th>STRATEGY</th>
<th>TEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Click the **Visualization** tab. If necessary, format the results as a line chart:

Each line represents a different type of product that is sold at the Buttercup Games online store. The height of each line shows the number of different people who bought the product during that hour. In general, it looks like the most popular items at the online shop were Arcade games.

You can format the report as a stacked column chart, which will show you the total purchases at each hour of the day.
1. Change the chart type to a **Column Chart**.
2. Use the **Format** menu, and on the **General** tab select **stacked**.

6. **Chart the number of earthquakes and the magnitude of each earthquake**

This example uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the **USGS Earthquake Feeds** and add it as an input.

Create a chart that list the number of earthquakes, and the magnitude of each earthquake that occurred in and around Alaska. Run the search using the time range **All time**.

```
source=all_month.csv place=*alaska* mag>=3.5 | chart count BY mag place
useother=f | rename mag AS Magnitude
```

This search counts the number of earthquakes that occurred in the Alaska regions. The count is then broken down for each **place** based on the magnitude of the quake. Because the **place** value is non-numeric, the search uses the **useother=f** argument to exclude events that don’t match.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>Magnitude</th>
<th>145km ENE of Chirikof Island, Alaska</th>
<th>225km SE of Kodiak, Alaska</th>
<th>250km SE of Kodiak, Alaska</th>
<th>252km SE of Kodiak, Alaska</th>
<th>254km SE of Kodiak, Alaska</th>
<th>255km SE of Kodiak, Alaska</th>
<th>259km SE of Kodiak, Alaska</th>
<th>264km SE of Kodiak, Alaska</th>
<th>265km SE of Kodiak, Alaska</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3.6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3.7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

253
<table>
<thead>
<tr>
<th>Magnitude</th>
<th>145km ENE of Chirikof Island, Alaska</th>
<th>225km SE of Kodiak, Alaska</th>
<th>250km SE of Kodiak, Alaska</th>
<th>252km SE of Kodiak, Alaska</th>
<th>254km SE of Kodiak, Alaska</th>
<th>255km SE of Kodiak, Alaska</th>
<th>259km SE of Kodiak, Alaska</th>
<th>264km SE of Kodiak, Alaska</th>
<th>265km SE of Kodiak, Alaska</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3.9</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4.6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Click on the **Visualization** tab to view the results on a chart. This chart shows the number of earthquakes by magnitude.
See also

Commands
  timechart
  bin
  sichart

Blogs
  Search commands > stats, chart, and timechart

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the chart command.

cluster

Description

The cluster command groups events together based on how similar they are to each other. Unless you specify a different field, cluster groups events based on the contents of the _raw field. The default grouping method is to break down the events into terms (match=term list) and compute the vector between events. Set a higher threshold value for t, if you want the command to be more discriminating about which events are grouped together.

The result of the cluster command appends two new fields to each event. You can specify what to name these fields with the countfield and labelfield parameters, which default to cluster_count and cluster_label. The cluster_count value is the number of events that are part of the cluster, or the cluster size. Each event in the cluster is assigned the cluster_label value of the cluster it belongs to. For example, if the search returns 10 clusters, then the clusters are labeled from 1 to 10.

Syntax

cluster [slc-options]...
Optional arguments

slc-options

Syntax: slc-options

Description: Options for configuring simple log clusters (slc).

SLC options

t

Syntax: \texttt{t=<num>}

Description: Sets the cluster threshold, which controls the sensitivity of the clustering. This value needs to be a number greater than 0.0 and less than 1.0. The closer the threshold is to 1, the more similar events have to be for them to be considered in the same cluster.

Default: 0.8

delims

Syntax: \texttt{delims=<string>}

Description: Configures the set of delimiters used to tokenize the raw string. By default, everything except 0-9, A-Z, a-z, and '_' are delimiters.

showcount

Syntax: \texttt{showcount=<bool>}

Description: If showcount=false, indexers cluster its own events before clustering on the search head. When showcount=false the event count is not added to the event. When showcount=true, the event count for each cluster is recorded and each event is annotated with the count.

Default: showcount=false

countfield

Syntax: \texttt{countfield=<field>}

Description: Name of the field to which the cluster size is to be written if showcount=true is true. The cluster size is the count of events in the cluster.

Default: \texttt{cluster\_count}.

labelfield

Syntax: \texttt{labelfield=<field>}

Description: Name of the field to write the cluster number to. As the events are grouped into clusters, each cluster is counted and labelled with a number.
Default: cluster_label

field
   Syntax: field=<field>
   Description: Name of the field to analyze in each event.
   Default: _raw

labelonly
   Description: labelonly=<bool>
   Syntax: Select whether to preserve incoming events and annotate them with the cluster they belong to (labelonly=true) or output only the cluster fields as new events (labelonly=false). When labelonly=false, outputs the list of clusters with the event that describes it and the count of events that combined with it.
   Default: false

match
   Syntax: match=(termlist | termset | ngramset)
   Description: Select the method used to determine the similarity between events. termlist breaks down the field into words and requires the exact same ordering of terms. termset allows for an unordered set of terms. ngramset compares sets of trigram (3-character substrings). ngramset is significantly slower on large field values and is most useful for short non-textual fields, like punct.
   Default: termlist

Usage

The cluster command is a streaming command or a dataset processing command, depending on which arguments are specified with the command. See Command types.

Use the cluster command to find common or rare events in your data. For example, if you are investigating an IT problem, use the cluster command to find anomalies. In this case, anomalous events are those that are not grouped into big clusters or clusters that contain few events. Or, if you are searching for errors, use the cluster command to see approximately how many different types of errors there are and what types of errors are common in your data.

Examples
Example 1

Quickly return a glimpse of anything that is going wrong in your Splunk deployment. Your role must have the appropriate capabilities to access the internal indexes.

```
index=_internal source=*splunkd.log* log_level!=info | cluster showcount=t | table cluster_count _raw | sort -cluster_count
```

This search takes advantage of what Splunk software logs about its operation in the _internal index. It returns all logs where the log_level is DEBUG, WARN, ERROR, FATAL and clusters them together. Then it sorts the clusters by the count of events in each cluster.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>cluster_count</th>
<th>raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>303010</td>
<td>03-20-2018 09:37:33.806 -0700 ERROR HotDBManager - Unable to create directory /Applications/Splunk/var/lib/splunk/_internaldb/db/hot_v1_49427345 because No such file or directory</td>
</tr>
<tr>
<td>151506</td>
<td>03-20-2018 09:37:33.811 -0700 ERROR pipeline - Uncaught exception in pipeline (indexer) - getting next event</td>
</tr>
<tr>
<td>16390</td>
<td>04-05-2018 08:30:53.996 -0700 WARN SearchResultsMem - Failed to append to multival. Original value not converted successfully to multival.</td>
</tr>
<tr>
<td>486</td>
<td>03-20-2018 09:37:33.814 -0700 ERROR BTreeCP - failed: failed to mkdir /Applications/Splunk/var/lib/splunk/fishbucket/splunk_private_db/snapshot.tmp: No such file or directory</td>
</tr>
<tr>
<td>216</td>
<td>03-20-2018 09:37:33.814 -0700 WARN DatabaseDirectoryManager - idx=_internal file=/Applications/Splunk/var/lib/splunk/_internaldb/db/.bucketManifest99454_1652919429_tmp for writing bucket manifest (No such file or directory)</td>
</tr>
<tr>
<td>216</td>
<td>03-20-2018 09:37:33.814 -0700 ERROR SearchResultsWriter - Unable to open output file: path=/Applications/Splunk/var/lib/splunk/_internaldb/db/.bucketManifest99454_1652919429_tmp error=No such file or directory</td>
</tr>
</tbody>
</table>

Example 2

Search for events that don't cluster into large groups.

```
... | cluster showcount=t | sort cluster_count
```

This returns clusters of events and uses the sort command to display them in ascending order based on the cluster size, which are the values of
cluster_count. Because they don’t cluster into large groups, you can consider these rare or uncommon events.

**Example 3**

Cluster similar error events together and search for the most frequent type of error.

```
error | cluster t=0.9 showcount=t | sort - cluster_count | head 20
```

This searches your index for events that include the term "error" and clusters them together if they are similar. The sort command is used to display the events in descending order based on the cluster size, cluster_count, so that largest clusters are shown first. The head command is then used to show the twenty largest clusters. Now that you’ve found the most common types of errors in your data, you can dig deeper to find the root causes of these errors.

**Example 4**

Use the cluster command to see an overview of your data. If you have a large volume of data, run the following search over a small time range, such as 15 minutes or 1 hour, or restrict it to a source type or index.

```
... | cluster labelonly=t showcount=t | sort - cluster_count, cluster_label, _time | dedup 5 cluster_label
```

This search helps you to learn more about your data by grouping events together based on their similarity and showing you a few of events from each cluster. It uses labelonly=t to keep each event in the cluster and append them with a cluster_label. The sort command is used to show the results in descending order by its size (cluster_count), then its cluster_label, then the indexed timestamp of the event (_time). The dedup command is then used to show the first five events in each cluster, using the cluster_label to differentiate between each cluster.

**See also**

anomalies, anomalousvalue, kmeans, outlier

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the cluster command.
cofilter

Description

Use this command to determine how many times field1 and field2 values occur together.

This command implements one step in a collaborative filtering analysis for making recommendations. Given a user field (field1) and an item field (field2), it finds how common each pair of items is. That is, it computes sum(A has X and A has Y) where X and Y are distinct items and A is each distinct user.

Syntax

cofilter <field1> <field2>

Required arguments

field1
Syntax: <field>
Description: The name of field.

field2
Syntax: <field>
Description: The name of a field.

Usage

The cofilter command is a transforming command. See Command types.

Examples

Example 1:

Find the cofilter for user and item. The user field must be specified first and followed by the item field. The output is event for each pair of items with: the first item and its popularity, the second item and its popularity, and the popularity of that pair of items.

... | cofilter user item
See also
associate, correlate

Answers
Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the cofilter command.

collect

Description
Adds the results of a search to a summary index that you specify. You must create the summary index before you invoke the collect command.

Syntax
collect index=<string> [<arg-options>...]

Required arguments

index

Syntax: index=<string>
Description: Name of the summary index where the events are added. The index must exist before the events are added. The index is not created automatically.

Optional arguments

arg-options

Syntax: addtime=<bool> | file=<string> | spool=<bool> | marker=<string> | testmode=<bool> | run_in_preview=<bool> | host=<string> | source=<string> | sourcetype=<string>
Description: Optional arguments for the collect command. See the arg-options section for the descriptions for each option.

arg-options

addtime

Syntax: addtime=<bool>
Description: Use this option to specify whether to prefix a time field on to each event. Some commands return results that do not have a _raw field, such as the stats, chart, timechart commands. If you specify addtime=false, the Splunk software uses its generic date detection against fields in whatever order they happen to be in the summary rows. If you specify addtime=true, the Splunk software uses the search time range info_min_time. This time range is added by the sistats) command or _time. Splunk software adds the time field based on the first field that it finds: info_min_time, _time, or now().
Default: true

file
Syntax: file=<string>
Description: The file name where you want the events to be written. You can use a timestamp or a random number for the file name by specifying either file=$timestamp$ or file=$random$.
Usage: ".stash" needs to be added at the end of the file name when used with "index=". Otherwise, the data is added to the main index.
Default: <random-number>_events.stash

host
Syntax: host=<string>
Description: The name of the host that you want to specify for the events.

marker
Syntax: marker=<string>
Description: A string, usually of key-value pairs, to append to each event written out. Each key-value pair must be separated by a comma and a space.

If the value contains spaces or commas, it must be escape quoted. For example if the key-value pair is search_name=vpn starts and stops, you must change it to search_name="vpn starts and stops".

run_in_preview
Syntax: run_in_preview=<bool>
Description: Controls whether the collect command is enabled during preview generation. Generally, you do not want to insert preview results into the summary index, run-in-preview=false. In some cases, such as when a custom search command is used as part of the search, you might want to turn this on to ensure correct summary indexable previews are generated.
Default: false

262
spool

**Syntax:** spool=<bool>

**Description:** If set to true, the summary indexing file is written to the Splunk spool directory, where it is indexed automatically. If set to false, the file is written to the `$SPLUNK_HOME/var/run/splunk` directory. The file remains in this directory unless some form of further automation or administration is done. If you have Splunk Enterprise, you can use this command to troubleshoot summary indexing by dumping the output file to a location on disk where it will not be ingested as data.

**Default:** true

source

**Syntax:** source=<string>

**Description:** The name of the source that you want to specify for the events.

sourcetype

**Syntax:** sourcetype=<string>

**Description:** The name of the source type that you want to specify for the events. By specifying a sourcetype outside of stash, you **will incur license usage**.

**Default:** stash

testmode

**Syntax:** testmode=<bool>

**Description:** Toggle between testing and real mode. In testing mode the results are not written into the new index but the search results are modified to appear as they would if sent to the index.

**Default:** false

**Usage**

The events are written to a file whose name format is: `random-num_events.stash`, unless overwritten, in a directory that your Splunk deployment is monitoring. If the events contain a `_raw` field, then this field is saved. If the events do not have a `_raw` field, one is created by concatenating all the fields into a comma-separated list of key=value pairs.

The `collect` command also works with real-time searches that have a time range of **All time**.
**Events without timestamps**

If you apply the `collect` command to events that do not have timestamps, the command designates a time for all of the events using the earliest (or minimum) time of the search range. For example, if you use the `collect` command over the past four hours (range: -4h to +0h), the command assigns a timestamp that is four hours prior to the time that the search was launched. The timestamp is applied to all of the events without a timestamp.

If you use the `collect` command with a time range of **All time** and the events do not have timestamps, the current system time is used for the timestamps.

For more information on summary indexing of data without timestamps, see "Use summary indexing for increased reporting efficiency" in the *Knowledge Manager Manual*.

**Copying events to a different index**

You can use the `collect` command to copy search results to another index. Construct a search that returns the data you want to copy, and pipe the results to the `collect` command. For example:

```sh
index=foo | ... | collect index=bar
```

This search writes the results into the `bar` index. The sourcetype is changed to `stash`.

You can specify a sourcetype with the `collect` command. However, specifying a sourcetype counts against your license, as if you indexed the data again.

**Examples**

1. **Put "download" events into an index named "download count"**

   ```sh
eventtypetag="download" | collect index=downloadcount
```

2. **Collect statistics on VPN connects and disconnects**

   You want to collect hourly statistics on VPN connects and disconnects by country.

   ```sh
   index=mysummary | geoip REMOTE_IP | eval country_source=if(REMOTE_IP_country_code="US","domestic","foreign") | bin _time span=1h | stats count by _time, vpn_action, country_source
   ```
addinfo | collect index=mysummary marker="summary_type=vpn,
summary_span=3600, summary_method=bin, search_name="vpn starts and
stops"

The addinfo command ensures that the search results contain fields that specify when the search was run to populate these particular index values.

See also

overlap, sichart, sirare, sistats, sitop, sitimechart, tscollect

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the collect command.

concurrency

Description

Concurrency measures the number of events which have spans that overlap with the start of each event. Alternatively, this measurement represents the total number of events in progress at the time that each particular event started, including the event itself. This command does not measure the total number of events that a particular event overlapped with during its total span.

Syntax

concurrency duration=<field> [start=<field>] [output=<field>]

Required arguments

duration

Syntax: duration=<field>
Description: A field that represents a span of time. This field must be a numeric with the same units as the start field. For example, the duration field generated by the transaction command is in seconds (see Example 1), which can be used with the default of _time which is also in units of seconds.
Optional arguments

start
Syntax: start=<field>
Description: A field that represents the start time.
Default: _time

output
Syntax: output=<field>
Description: A field to write the resulting number of concurrent events.
Default: "concurrency"

Usage

An event X is concurrent with event Y if X.start is between Y.start and (Y.start + Y.duration)

If your events have a time that represents event completion and a span that represents the time before the completion, you need to subtract duration from the start time before the concurrency command:

... |eval new_start = start - duration | concurrency start=new_start duration=duration

Limits

There is a limitation on quantity of overlapping items. If the maximum tracked concurrency exceeds max_count, from the [concurrency] stanza in limits.conf, a warning will be produced in the UI / search output, and the values will be clamped, making them potentially inaccurate. This limit defaults to 10000000 or ten million.

Basic examples

1. Determine the number of overlapping HTTP requests

Determine the number of overlapping HTTP requests outstanding from browsers accessing splunkd at the time that each http request begins.

This relies on the fact that the timestamp of the logged message is the time that the request came in, and the 'spent' field is the number of milliseconds spent handling the request. As always, you must be an 'admin' user, or have altered your roles scheme in order to access the _internal index.
index=_internal sourcetype=splunkd_ui_access | eval spent_in_seconds = spent / 1000 | concurrency duration=spent_in_seconds

2. **Calculate the number of concurrent events**

Calculate the number of concurrent events for each event and emit as field 'foo':

```
... | concurrency duration=total_time output=foo
```

3. **Use existing fields to specify the start time and duration**

Calculate the number of concurrent events using the 'et' field as the start time and 'length' as the duration:

```
... | concurrency duration=length start=et
```

**Extended examples**

1. **Count the transactions that occurred at the same time**

   This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range **All time** when you run the search.

   Use the duration or span of a transaction to count the number of other transactions that occurred at the same time.

   ```
sourcetype=access_* | transaction JSESSIONID clientip startswith="view" endswith="purchase" | concurrency duration=length | eval duration=tostring(duration,"duration")
   ```

   - This search groups events into transactions if they have the same values of JSESSIONID and clientip. An event is the beginning of the transaction if the event contains the string "view". An event is the last event of the transaction if the event contains the string "purchase".
   - The transaction command returns a field called **duration**.
   - The transactions are then piped into the concurrency command, which counts the number of events that occurred at the same time based on the timestamp and duration of the transaction.
   - The search also uses the eval command and the tostring() function to reformat the values of the duration field to a more readable format, HH:MM:SS.
To see the values in each transaction for the JSESSIONID, clientip, concurrency, and duration fields:

1. In the list of Interesting Fields, click the field name.
2. In the information box, for **Selected**, click **Yes**.
3. Select the next field in the list of Interesting Fields. The information box automatically refreshes. For **Selected**, click **Yes**.
4. Repeat these steps for every field you want to appear in the result list. The results should appear similar to the following image:

2. **Count the purchases that occurred at the same time**

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to **get the tutorial data into Splunk**. Use the time range **All time** when you run the search.

Use the time between each purchase to count the number of different purchases that occurred at the same time.
This search uses the `delta` command and the `_time` field to calculate the time between one purchase event (`action=purchase`) and the purchase event immediately preceding it.

- The search renames this change in time as `timeDelta`.
- Some of the values of `timeDelta` are negative. Because the `concurrency` command does not work with negative values, the `eval` command is used to redefine `timeDelta` as its absolute value (`abs(timeDelta)`).
- The `timeDelta` is then used as the `duration` for calculating concurrent events.

3. Calculate the transactions using the time between consecutive transactions

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range `All time` when you run the search.

Use the time between each consecutive transaction to calculate the number of transactions that occurred at the same time.

- This search groups events into transactions if they have the same values of `JSESSIONID` and `clientip`. An event is the beginning of the transaction if the event contains the string "view". An event is the last event of the transaction if the event contains the string "purchase".
• The `transaction` command returns a field called **duration**.
• The transactions are then piped into the `delta` command, which uses the `_time` field to calculate the time between one transaction and the transaction immediately preceding it.
• The search renames this change in time as **timeDelta**.
• Some of the values of `timeDelta` are negative. Because the `concurrency` command does not work with negative values, the `eval` command is used to redefine `timeDelta` as its absolute value (`abs(timeDelta)`).
• This `timeDelta` is then used as the `duration` for calculating concurrent transactions.

See also

timechart

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the concurrency command.

contingency

Description

In statistics, contingency tables are used to record and analyze the relationship between two or more (usually categorical) variables. Many metrics of association or independence, such as the **phi coefficient** or the **Cramer's V**, can be calculated.
based on contingency tables.

You can use the `contingency` command to build a contingency table, which in this case is a co-occurrence matrix for the values of two fields in your data. Each cell in the matrix displays the count of events in which both of the cross-tabulated field values exist. This means that the first row and column of this table is made up of values of the two fields. Each cell in the table contains a number that represents the count of events that contain the two values of the field in that row and column combination.

If a relationship or pattern exists between the two fields, you can spot it easily just by analyzing the information in the table. For example, if the column values vary significantly between rows (or vice versa), there is a contingency between the two fields (they are not independent). If there is no contingency, then the two fields are independent.

**Syntax**

`contingency [contingency-options...] <field1> <field2>`

**Required arguments**

`<field1>`

- **Syntax:** `<field>`
- **Description:** Any field. You cannot specify wildcard characters in the field name.

`<field2>`

- **Syntax:** `<field>`
- **Description:** Any field. You cannot specify wildcard characters in the field name.

**Optional arguments**

`contingency-options`

- **Syntax:** `<maxopts> | <mincover> | <usetotal> | <totalstr>`
- **Description:** Options for the contingency table.
**Contingency options**

**maxopts**

**Syntax:** `maxrows=<int> | maxcols=<int>`

**Description:** Specify the maximum number of rows or columns to display. If the number of distinct values of the field exceeds this maximum, the least common values are ignored. A value of 0 means a maximum limit on rows or columns. This limit comes from the `maxvalues` setting in the `[ctable]` stanza in the `limits.conf` file.

**Default:** 1000

**mincover**

**Syntax:** `mincolcover=<num> | minrowcover=<num>`

**Description:** Specify a percentage of values per column or row that you would like represented in the output table. As the table is constructed, enough rows or columns are included to reach this ratio of displayed values to total values for each row or column. The maximum rows or columns take precedence if those values are reached.

**Default:** 1.0

**usetotal**

**Syntax:** `usetotal=<bool>`

**Description:** Specify whether or not to add row, column, and complete totals.

**Default:** true

**totalstr**

**Syntax:** `totalstr=<field>`

**Description:** Field name for the totals row and column.

**Default:** TOTAL

**Usage**

The **contingency** command is a **transforming command**. See Command types.

This command builds a contingency table for two fields. If you have fields with many values, you can restrict the number of rows and columns using the `maxrows` and `maxcols` arguments.

**Totals**

By default, the contingency table displays the row totals, column totals, and a grand total for the counts of events that are represented in the table. If you don’t
want the totals to appear in the results, include the `usetotal=false` argument with the `contingency` command.

**Empty values**

Values which are empty strings (""") will be represented in the results table as `EMPTY_STR`.

**Limits**

There is a limit on the value of `maxrows` or `maxcols`, which means more than 1000 values for either field will not be used.

**Examples**

1. **Build a contingency table of recent data**

   ```
   This search uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

   You can download a current CSV file from the USGS Earthquake Feeds and upload the file to your Splunk instance. This example uses the All Earthquakes data from the past 30 days. Use the time range All time when you run the searches.
   ```

You want to build a contingency table to look at the relationship between the magnitudes and depths of recent earthquakes. You start with a simple search.

   ```
   source=all_month.csv | contingency mag depth | sort mag
   ```

   There are quite a range of values for the Magnitude and Depth fields, which results in a very large table. The magnitude values appear in the first column. The depth values appear in the first row. The list is sorted by magnitude.

   The results appear on the Statistics tab. The following table shows only a small portion of the table of results returned from the search.

<table>
<thead>
<tr>
<th>mag</th>
<th>10</th>
<th>0</th>
<th>5</th>
<th>35</th>
<th>8</th>
<th>12</th>
<th>15</th>
<th>11.9</th>
<th>11.8</th>
<th>6.4</th>
<th>5.4</th>
<th>8.2</th>
<th>6.5</th>
<th>8.1</th>
<th>5.6</th>
<th>10.1</th>
<th>9</th>
<th>8.5</th>
<th>9.8</th>
<th>8.7</th>
<th>7.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.81</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-0.59</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
As you can see, earthquakes can have negative magnitudes. Only where an earthquake occurred that matches the magnitude and depth will a count appear in the table.

To build a more usable contingency table, you should reformat the values for the magnitude and depth fields. Group the magnitudes and depths into ranges.

<table>
<thead>
<tr>
<th>Name</th>
<th>Shallow</th>
<th>Mid</th>
<th>Deep</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 - 1.0</td>
<td>3579</td>
<td>33</td>
<td>0</td>
<td>3612</td>
</tr>
<tr>
<td>1.1 - 2.0</td>
<td>3188</td>
<td>596</td>
<td>0</td>
<td>3784</td>
</tr>
<tr>
<td>2.1 - 3.0</td>
<td>1236</td>
<td>131</td>
<td>0</td>
<td>1367</td>
</tr>
<tr>
<td>3.1 - 4.0</td>
<td>320</td>
<td>63</td>
<td>1</td>
<td>384</td>
</tr>
<tr>
<td>4.1 - 5.0</td>
<td>400</td>
<td>157</td>
<td>43</td>
<td>600</td>
</tr>
<tr>
<td>5.1 - 6.0</td>
<td>63</td>
<td>12</td>
<td>3</td>
<td>78</td>
</tr>
<tr>
<td>6.1 - 7.0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>TOTAL</td>
<td>8788</td>
<td>994</td>
<td>48</td>
<td>9830</td>
</tr>
</tbody>
</table>

There were a lot of quakes in this month. Do higher magnitude earthquakes have a greater depth than lower magnitude earthquakes? Not really. The table shows
that the majority of the recent earthquakes in all of magnitude ranges were shallow. There are significantly fewer earthquakes in the mid-to-deep range. In this data set, the deep-focused quakes were all in the mid-range of magnitudes.

2. Identify potential component issues in the Splunk deployment

Determine if there are any components that might be causing issues in your Splunk deployment. Build a contingency table to see if there is a relationship between the values of log_level and component. Run the search using the time range All time and limit the number of columns returned.

index=_internal | contingency maxcols=5 log_level component

Your results should appear something like this:

These results show you any components that might be causing issues in your Splunk deployment. The component field has more than 50 values. In this search, the maxcols argument is used to show 5 components with the highest values.

See also

associate, correlate

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the contingency command.
Description

The `convert` command converts field values in your search results into numerical values. Unless you use the AS clause, the original values are replaced by the new values.

Alternatively, you can use evaluation functions such as `strftime()`, `strptime()`, or `tonumber()` to convert field values.

Syntax

```
convert [timeformat=string] (<convert-function> [AS <field>])...
```

**Required arguments**

`<convert-function>`

**Syntax:** `auto() | ctime() | dur2sec() | memk() | mktime() | mstime() | none() | num() | rmcomma() | rmunit()

**Description:** Functions to use for the conversion.

**Optional arguments**

`timeformat`

**Syntax:** `timeformat=<string>

**Description:** Specify the output format for the converted time field. The `timeformat` option is used by `ctime` and `mktime` functions. For a list and descriptions of format options, see Common time format variables in the Search Reference.

**Default:** `%m/%d/%Y %H:%M:%S. Note that this default does not conform to the locale settings.

`<field>`

**Syntax:** `<string>

**Description:** Creates a new field with the name you specify to place the converted values into. The original field and values remain intact.

Convert functions

`auto()`

**Syntax:** `auto(<wc-field>)`

**Description:** Automatically convert the fields to a number using the best conversion. Note that if not all values of a particular field can be converted using a known conversion type, the field is left untouched and no
conversion at all is done for that field. You can use wild card characters in
the field name.

cTIME()  
**Syntax:** ctime(<wc-field>)  
**Description:** Convert an epoch time to an ascii human readable time. Use
the timeformat option to specify exact format to convert to. You can use
wild card characters in the field name.

dur2sec()  
**Syntax:** dur2sec(<wc-field>)  
**Description:** Convert a duration format "[D+]HH:MM:SS" to seconds. You
can use wild card characters in the field name.

memk()  
**Syntax:** memk(<wc-field>)  
**Description:** Accepts a positive number (integer or float) followed by an
optional "k", "m", or "g". The letter k indicates kilobytes, m indicates
megabytes, and g indicates gigabytes. If no letter is specified, kilobytes is
assumed. The output field is a number expressing quantity of kilobytes.
Negative values cause data incoherency. You can use wild card
characters in the field name.

mktime()  
**Syntax:** mktime(<wc-field>)  
**Description:** Convert a human readable time string to an epoch time. Use
timeformat option to specify exact format to convert from. You can use
wild card characters in the field name.

mstime()  
**Syntax:** mstime(<wc-field>)  
**Description:** Convert a [MM:]SS.SSS format to seconds. You can use
wild card characters in the field name.

none()  
**Syntax:** none(<wc-field>)  
**Description:** In the presence of other wildcards, indicates that the
matching fields should not be converted. You can use wild card characters
in the field name.

num()  
**Syntax:** num(<wc-field>)
Description: Like auto(), except non-convertible values are removed. You can use wild card characters in the field name.

rmcomma()
Syntax: rmcomma(<wc-field>)
Description: Removes all commas from value, for example rmcomma(1,000,000.00) returns 1000000.00. You can use wild card characters in the field name.

rmunit()
Syntax: rmunit(<wc-field>)
Description: Looks for numbers at the beginning of the value and removes trailing text. You can use wild card characters in the field name.

Usage

The convert command is a distributable streaming command. See Command types.

Basic examples

1. Convert all field values to numeric values

Use the auto convert function to convert all field values to numeric values.

... | convert auto(*)

2. Convert field values except for values in specified fields

Convert every field value to a number value except for values in the field foo. Use the none convert function to specify fields to ignore.

... | convert auto(*) none(foo)

3. Change the duration values to seconds for the specified fields

Change the duration values to seconds for the specified fields

... | convert dur2sec(xdelay) dur2sec(delay)
4. Change the sendmail syslog duration format to seconds

Change the sendmail syslog duration format (D+HH:MM:SS) to seconds. For example, if `delay="00:10:15"`, the resulting value is `delay="615"`. This example uses the `dur2sec` convert function.

```
... | convert dur2sec(delay)
```

5. Convert field values that contain numeric and string values

Convert the values in the `duration` field, which contain numeric and string values, to numeric values by removing the string portion of the values. For example, if `duration="212 sec"`, the resulting value is `duration="212"`. This example uses the `rmunit` convert function.

```
... | convert rmunit(duration)
```

6. Change memory values to kilobytes

Change all memory values in the `virt` field to KBs. This example uses the `memk` convert function.

```
... | convert memk(virt)
```

Extended Examples

1. Convert a UNIX time to a more readable time format

Convert a UNIX time to a more readable time formatted to show hours, minutes, and seconds.

```
source="all_month.csv" | convert timeformat="%H:%M:%S" ctime(_time) AS c_time | table _time, c_time
```

- The `ctime()` function converts the `_time` value in the CSV file events to the format specified by the `timeformat` argument.
- The `timeformat="%H:%M:%S"` argument tells the search to format the `_time` value as HH:MM:SS.
- The converted time `ctime` field is renamed `c_time`.
- The `table` command is used to show the original `_time` value and the `ctime` field.

The results appear on the Statistics tab and look something like this:
The `ctime()` function changes the timestamp to a non-numerical value. This is useful for display in a report or for readability in your events list.

### 2. Convert a time in MM:SS.SSS to a number in seconds

Convert a time in MM:SS.SSS (minutes, seconds, and subseconds) to a number in seconds.

```
sourcetype=syslog | convert mstime(_time) AS ms_time | table _time, ms_time
```

- The `mstime()` function converts the `_time` field values from a minutes and seconds to just seconds.

The converted time field is renamed `ms_time`.

- The `table` command is used to show the original `_time` value and the converted time.

<table>
<thead>
<tr>
<th>_time</th>
<th>ms_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-03-27 17:20:14.839</td>
<td>1522196414.839</td>
</tr>
<tr>
<td>2018-03-27 17:21:05.724</td>
<td>1522196465.724</td>
</tr>
<tr>
<td>2018-03-27 17:27:03.790</td>
<td>1522196823.790</td>
</tr>
<tr>
<td>2018-03-27 17:28:41.869</td>
<td>1522196921.869</td>
</tr>
<tr>
<td>2018-03-27 17:34:40.900</td>
<td>1522197280.900</td>
</tr>
<tr>
<td>2018-03-27 17:38:47.120</td>
<td>1522197527.120</td>
</tr>
<tr>
<td>2018-03-27 17:40:10.345</td>
<td>1522197610.345</td>
</tr>
</tbody>
</table>
The `mstime()` function changes the timestamp to a numerical value. This is useful if you want to use it for more calculations.

3. **Convert a string time in HH:MM:SS into a number**

Convert a string field `time_elapsed` that contains times in the format HH:MM:SS into a number. Sum the `time_elapsed` by the `user_id` field. This example uses the `eval` command to convert the converted results from seconds into minutes.

```bash
...| convert num(time_elapsed) | stats sum(eval(time_elapsed/60)) AS Minutes BY user_id
```

### See also

**Commands**
- `eval`
- `fieldformat`

**Functions**
- `tonumber`
- `strptime`

### Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the convert command.

### correlate

**Description**

Calculates the correlation between different fields.

You can use the `correlate` command to see an overview of the co-occurrence between fields in your data. The results are presented in a matrix format, where the cross tabulation of two fields is a cell value. The cell value represents the percentage of times that the two fields exist in the same events.
The field the result is specific to is named in the value of the RowField field, while the fields it is compared against are the names of the other fields.

**Note:** This command looks at the relationship among all the fields in a set of search results. If you want to analyze the relationship between the values of fields, refer to the contingency command, which counts the co-occurrence of pairs of field values in events.

**Syntax**

correlate

**Limits**

There is a limit on the number of fields that correlate considers in a search. From limits.conf, stanza [correlate], the maxfields sets this ceiling. The default is 1000.

If more than this many fields are encountered, the correlate command continues to process data for the first N (eg thousand) field names encountered, but ignores data for additional fields. If this occurs, the notification from the search or alert contains a message "correlate: input fields limit (N) reached. Some fields may have been ignored."

As with all designed-in limits, adjusting this might have significant memory or cpu costs.

**Examples**

**Example 1:**

Look at the co-occurrence between all fields in the _internal index.

index=_internal | correlate

Here is a snapshot of the results.
Because there are different types of logs in the _internal, you can expect to see that many of the fields do not co-occur.

**Example 2:**

Calculate the co-occurrences between all fields in Web access events.

```
sourcetype=access_*  | correlate
```

You expect all Web access events to share the same fields: clientip, referer, method, and so on. But, because the `sourcetype=access_*` includes both access_common and access_combined Apache log formats, you should see that the percentages of some of the fields are less than 1.0.

**Example 3:**

Calculate the co-occurrences between all the fields in download events.

```
eventtype=download  | correlate
```

The more narrow your search is before you pass the results into `correlate`, the more likely it is that all the field value pairs have a correlation of 1.0. A correlation of 1.0 means the values co-occur in 100% of the search results. For these download events, you might be able to spot an issue depending on which pairs have less than 1.0 co-occurrence.

**See also**

`associate, contingency`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `correlate` command.

**ctable**

The `ctable`, or `counttable`, command is an alias for the `contingency` command. See the `contingency` command for the syntax and examples.
See also

associates, correlate

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the ctable command.

datamodel

Description

Examine and search data model datasets.

Use the datamodel command to return the JSON for all or a specified data model and its datasets. You can also search against the specified data model or a dataset within that datamodel.

A data model is a hierarchically-structured search-time mapping of semantic knowledge about one or more datasets. A data model encodes the domain knowledge necessary to build a variety of specialized searches of those datasets. For more information, see About data models and Design data models in the Knowledge Manager Manual.

The datamodel search command lets you search existing data models and their datasets from the search interface.

The datamodel command is a generating command and should be the first command in the search. Generating commands use a leading pipe character.

Syntax

| datamodel [<data model name>] [<dataset name>] [<data model search mode>] [strict_fields=<bool>] [allow_old_summaries=<bool>] [summariesonly=<bool>]
**Required arguments**

None

**Optional arguments**

data model name  
**Syntax:** <string>  
**Description:** The name of the data model to search. When only the data model is specified, the search returns the JSON for the single data model.

dataset name  
**Syntax:** <string>  
**Description:** The name of a data model dataset to search. Must be specified after the data model name. The search returns the JSON for the single dataset.

data model search mode  
**Syntax:** <data model search result mode> | <data model search string mode>  
**Description:** You can use `datamodel` to run a search against a data model or a data model dataset that returns either results or a search string. If you want to do this, you must provide a `<data model search mode>`. There are two `<data model search mode>` subcategories: modes that return results and modes that return search strings. See `<data model search mode>` options.

allow_old_summaries  
**Syntax:** allow_old_summaries=<bool>  
**Description:** This argument applies only to accelerated data models. When you change the constraints that define a data model but the Splunk software has not fully updated the summaries to reflect that change, the summaries may have some data that matches the old definition and some data that matches the new definition. By default, `allow_old_summaries = false`, which means that the search head does not use summary directories that are older than the new summary definition. This ensures that the `datamodel` search results always reflect your current configuration. When you set `allow_old_summaries = false`, `datamodel` uses both current summary data and summary data that was generated prior to the definition change. You can set `allow_old_summaries=true` in your search if you feel that the old summary data is close enough to the new summary data that its results are reliable.  
**Default:** false
summariesonly

Syntax: summariesonly=<bool>

Description: This argument applies only to accelerated data models. When set to false, the datamodel search returns both summarized and unsummarized data for the selected data model. When set to true, the search returns results only from the data that has been summarized in TSIDX format for the selected data model. You can use this argument to identify what data is currently summarized for a given data model, or to ensure that a particular data model search runs efficiently.

Default: false

strict_fields

Syntax: strict_fields=<bool>

Description: Determines the scope of the datamodel search in terms of fields returned. When strict_fields=true, the search returns only default fields and fields that are included in the constraints of the specified data model dataset. When strict_fields=false, the search returns all fields defined in the data model, including fields inherited from parent data model datasets, extracted fields, calculated fields, and fields derived from lookups.

You can also arrange for strict_fields to default to false for a specific data model. See Design data models in the Knowledge Manager Manual.

Default: true

<data model search mode> options

data model search result mode

Syntax: search | flat | acceleration_search

Description: The modes for running searches on a data model or data model dataset that return results.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>Returns the search results exactly how they are defined.</td>
</tr>
<tr>
<td>flat</td>
<td>Returns the same results as the search, except that it strips the hierarchical information from the field names. For example, where search mode might return a field named dmdataset.server, the field mode returns a field named server.</td>
</tr>
<tr>
<td>acceleration_search</td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>data model search string mode</td>
<td>Runs the search that the search head uses to accelerate the data model. This mode works only on root event datasets and root search datasets that only use streaming commands.</td>
</tr>
</tbody>
</table>

**data model search string mode**

**Syntax:** `search_string | flat_string | acceleration_search_string`

**Description:** These modes return the strings for the searches that the Splunk software is actually running against the data model when it runs your SPL through the corresponding `<data model search result mode>`. For example, if you choose `acceleration_search_string`, the Splunk software returns the search string it would actually use against the data model when you run your SPL through `acceleration_search` mode.

**Usage**

The `datamodel` command is a report-generating command. See **Command types**.

Generating commands use a leading pipe character and should be the first command in a search.

**Examples**

1. **Return the JSON for all data models**

   Return JSON for all data models available in the current app context.

   ```bash
   | datamodel
   ```
2. Return the JSON for a specific datamodel

Return JSON for the **Splunk's Internal Audit Logs - SAMPLE** data model, which has the model ID `internal_audit_logs`.

```
| datamodel internal_audit_logs
```

3. Return the JSON for a specific dataset

Return JSON for Buttercup Games's Client_errors dataset.

```
| datamodel Tutorial Client_errors
```

4. Run a search on a specific dataset

Run the search for Buttercup Games's Client_errors.

```
| datamodel Tutorial Client_errors search
```
5. Run a search on a dataset for specific criteria

Search Buttercup Games’s Client_errors dataset for 404 errors and count the number of events.

```bash
| datamodel Tutorial Client_errors search | search Tutorial.status=404 | stats count
```

6. For an accelerated data model, reveal what data has been summarized over a selected time range

After the Tutorial data model is accelerated, this search uses the `summariesonly` argument in conjunction with `timechart` to reveal what data has been summarized for the Client_errors dataset over a selected time range.

```bash
| datamodel Tutorial summariesonly=true search | timechart span=1h count
```

See also

`pivot`

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the `datamodel` command.

**datamodelsimple**

The `datamodelsimple` command is used with the Splunk Common Information Model Add-on.

For information about this command, see Use the `datamodelsimple` command in the [Common Information Model Add-on Manual](#).

**dbinspect**

**Description**

Returns information about the buckets in the specified index. If you are using Splunk Enterprise, this command helps you understand where your data resides...
so you can optimize disk usage as required.

The Splunk **index** is the repository for data ingested by Splunk software. As incoming data is indexed and transformed into **events**, Splunk software creates files of **rawdata** and metadata (**index files**). The files reside in sets of directories organized by age. These directories are called **buckets**.

For more information, see Indexes, indexers, and clusters and How the indexer stores indexes in *Managing Indexers and Clusters of Indexers*.

**Syntax**

```
| dbinspect [index=<wc-string>]... [span | <timeformat>] [corruptonly=<bool>]
```

**Required arguments**

None.

**Optional arguments**

**index**

**Syntax:** index=<wc-string>...

**Description:** Specifies the name of an index to inspect. You can specify more than one index. For all non-internal indexes, you can specify an asterisk (* ) in the index name.

**Default:** The default index, which is typically main.

**<span>**

**Syntax:** span=<int> | span=<int><timescale>

**Description:** Specifies the span length of the bucket. If using a timescale unit (second, minute, hour, day, month, or subseconds), this is used as a time range. If not, this is an absolute bucket "length".

When you invoke the `dbinspect` command with a bucket span, a table of the spans of each bucket is returned. When `span` is not specified, information about the buckets in the index is returned. See Information returned when no bucket span is specified.

**<timeformat>**

**Syntax:** timeformat=<string>

**Description:** Sets the time format for the modTime field.

**Default:** timeformat=%m/%d/%Y:%H:%M:%S
**<corruptonly>**

**Syntax:** corruptonly=<bool>

**Description:** Specifies that each bucket is checked to determine if any buckets are corrupted and displays only the corrupted buckets. A bucket is corrupt when some of the files in the bucket are incorrect or missing such as Hosts.data or tsidx. Corrupt bucket might return incorrect data or render the bucket unsearchable. In most cases the software will auto-repair corrupt buckets.

When `corruptonly=true`, each bucket is checked and the following informational message appears.

INFO: The "corruptonly" option will check each of the specified buckets. This search might be slow and will take time.

**Default:** false

**Time scale units**

These are options for specifying a timescale as the bucket span.

**<timescale>**

**Syntax:** <sec> | <min> | <hr> | <day> | <month> | <subseconds>

**Description:** Time scale units.

<table>
<thead>
<tr>
<th>Time scale</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sec&gt;</td>
<td>s</td>
<td>sec</td>
</tr>
<tr>
<td>&lt;min&gt;</td>
<td>m</td>
<td>min</td>
</tr>
<tr>
<td>&lt;hr&gt;</td>
<td>h</td>
<td>hr</td>
</tr>
<tr>
<td>&lt;day&gt;</td>
<td>d</td>
<td>day</td>
</tr>
<tr>
<td>&lt;month&gt;</td>
<td>mon</td>
<td>month</td>
</tr>
<tr>
<td>&lt;subseconds&gt;</td>
<td>us</td>
<td>ms</td>
</tr>
</tbody>
</table>
When you invoke the `dbinspect` command without the `span` argument, the following information about the buckets in the index is returned.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bucketId</td>
<td>A string comprised of <code>&lt;index&gt;~&lt;id&gt;~&lt;guId&gt;</code>, where the delimiters are tilde characters. For example, <code>summary~2~4491025B-8E6D-48DA-A90E-89AC3CF2CE80</code>.</td>
</tr>
<tr>
<td>endEpoch</td>
<td>The timestamp for the last event in the bucket, which is the time-edge of the bucket furthest towards the future. Specify the timestamp in the number of seconds from the UNIX epoch.</td>
</tr>
<tr>
<td>eventCount</td>
<td>The number of events in the bucket.</td>
</tr>
<tr>
<td>guId</td>
<td>The globally unique identifier (GUID) of the server that hosts the index. This is relevant for index replication.</td>
</tr>
<tr>
<td>hostCount</td>
<td>The number of unique hosts in the bucket.</td>
</tr>
<tr>
<td>id</td>
<td>The local ID number of the bucket, generated on the indexer on which the bucket originated.</td>
</tr>
<tr>
<td>index</td>
<td>The name of the index specified in your search. You can specify <code>index=*</code> to inspect all of the indexes, and the index field will vary accordingly.</td>
</tr>
<tr>
<td>modTime</td>
<td>The timestamp for the last time the bucket was modified or updated, in a format specified by the <code>timeformat</code> flag.</td>
</tr>
<tr>
<td>path</td>
<td>The location to the bucket. The naming convention for the bucket path varies slightly, depending on whether the bucket rolled to warm while its indexer was functioning as a cluster peer:</td>
</tr>
<tr>
<td></td>
<td>• For non-clustered buckets: <code>db_&lt;newest_time&gt;_&lt;oldest_time&gt;_&lt;localid&gt;</code></td>
</tr>
<tr>
<td></td>
<td>• For clustered original bucket copies: <code>db_&lt;newest_time&gt;_&lt;oldest_time&gt;_&lt;localid&gt;_&lt;guid&gt;</code></td>
</tr>
<tr>
<td></td>
<td>• For clustered replicated bucket copies: <code>rb_&lt;newest_time&gt;_&lt;oldest_time&gt;_&lt;localid&gt;_&lt;guid&gt;</code></td>
</tr>
</tbody>
</table>

For more information, read "How Splunk stores indexes" and "Basic cluster architecture" in *Managing Indexers and Clusters of Indexers*. 

292
<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rawSize</td>
<td>The volume in bytes of the raw data files in each bucket. This value represents the volume before compression and the addition of index files.</td>
</tr>
<tr>
<td>sizeOnDiskMB</td>
<td>The size in MB of disk space that the bucket takes up expressed as a floating point number. This value represents the volume of the compressed raw data files and the index files.</td>
</tr>
<tr>
<td>sourceCount</td>
<td>The number of unique sources in the bucket.</td>
</tr>
<tr>
<td>sourceTypeCount</td>
<td>The number of unique sourcetypes in the bucket.</td>
</tr>
<tr>
<td>splunk_server</td>
<td>The name of the Splunk server that hosts the index in a distributed environment.</td>
</tr>
<tr>
<td>startEpoch</td>
<td>The timestamp for the first event in the bucket (the time-edge of the bucket furthest towards the past), in number of seconds from the UNIX epoch.</td>
</tr>
<tr>
<td>state</td>
<td>Whether the bucket is warm, hot, cold.</td>
</tr>
<tr>
<td>corruptReason</td>
<td>Specifies the reason why the bucket is corrupt. The corruptReason field appears only when corruptonly=true.</td>
</tr>
</tbody>
</table>

**Usage**

The `dbinspect` command is a generating command and should be the first command in the search. Generating commands use a leading pipe character.

**Accessing data and security**

If no data is returned from the index that you specify with the `dbinspect` command, it is possible that you do not have the authorization to access that index. The ability to access data in the Splunk indexes is controlled by the authorizations given to each role. See Use access control to secure Splunk data in *Securing Splunk Enterprise*.

**Non-searchable bucket copies**

For hot non-searchable bucket copies on target peers, tsidx and other metadata files are not maintained. Because accurate information cannot be reported, the following fields show NULL:

- eventCount
- hostCount
Examples

1. **CLI use of the `dbinspect` command**

Display a chart with the span size of 1 day, using the command line interface (CLI).

```
myLaptop $ splunk search "| dbinspect index=_internal span=1d"
```

```
<table>
<thead>
<tr>
<th>_time</th>
<th>hot-3</th>
<th>warm-1</th>
<th>warm-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-01-17 00:00:00.000 PST</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015-01-17 14:56:39.000 PST</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015-02-19 00:00:00.000 PST</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2015-02-20 00:00:00.000 PST</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```

2. **Default `dbinspect` output**

Default dbinspect output for a local `_internal` index.

```
| dbinspect index=_internal
```

```
This screen shot does not display all of the columns in the output table. On your computer, scroll to the right to see the other columns.

3. **Check for corrupt buckets**

Use the `corruptonly` argument to display information about corrupted buckets, instead of information about all buckets. The output fields that display are the same with or without the `corruptonly` argument.

```
| dbinspect index=_internal corruptonly=true
```

4. **Count the number of buckets for each Splunk server**

Use this command to verify that the Splunk servers in your distributed environment are included in the `dbinspect` command. Counts the number of buckets for each server.

```
| dbinspect index=_internal | stats count by splunk_server
```

5. **Find the index size of buckets in GB**

Use `dbinspect` to find the index size of buckets in GB. For current numbers, run this search over a recent time range.

```
| dbinspect index=_internal | eval GB=sizeOnDiskMB/1024 | stats sum(GB)
```

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `dbinspect` command.

**dedup**

**Description**

Removes the events that contain an identical combination of values for the fields that you specify.

With the `dedup` command, you can specify the number of duplicate events to keep for each value of a single field, or for each combination of values among several fields. Events returned by `dedup` are based on search order. For **historical searches**, the most recent events are searched first. For **real-time**
searches, the first events that are received are searched, which are not necessarily the most recent events.

You can specify the number of events with duplicate values, or value combinations, to keep. You can sort the fields, which determines which event is retained. Other options enable you to retain events with the duplicate fields removed, or to keep events where the fields specified do not exist in the events.

Syntax

dedup [<int>] <field-list> [keepevents=<bool>] [keepempty=<bool>] [consecutive=<bool>] [sortby <sort-by-clause>]

Required arguments

<field-list>

Syntax: <string> <string> ...
Description: A list of field names to remove duplicate values from.

Optional arguments

consecutive

Syntax: consecutive=<bool>
Description: If true, only remove events with duplicate combinations of values that are consecutive.
Default: false

keepempty

Syntax: keepempty=<bool>
Description: If set to true, keeps every event where one or more of the specified fields is not present (null).
Default: false. All events where any of the selected fields are null are dropped.

The keepempty=true argument keeps every event that does not have one or more of the fields in the field list. To keep N representative events for combinations of field values including null values, use the fillnull command to provide a non-null value for these fields. For example:

... | fillnull value="MISSING" field1 field2 | dedup field1 field2

keepevents

Syntax: kkeepevents=<bool>
Description: If true, keep all events, but will remove the selected fields from events after the first event containing a particular combination of values.
Default: false. Events are dropped after the first event of each particular combination.

<N>
Syntax: <int>
Description: The dedup command retains multiple events for each combination when you specify N. The number for N must be greater than 0. If you do not specify a number, only the first occurring event is kept. All other duplicates are removed from the results.

<sort-by-clause>
Syntax: sortby ( - | + ) <sort-field> [(- | +) <sort_field> ...]
Description: List of the fields to sort by and the sort order. Use the dash symbol ( - ) for descending order and the plus symbol ( + ) for ascending order. You must specify the sort order for each field specified in the <sort-by-clause>. The <sort-by-clause> determines which of the duplicate events to keep. When the list of events is sorted, the top-most event in the sorted list is retained.

Sort field options

<sort-field>
Syntax: <field> | auto(<field>) | str(<field>) | ip(<field>) | num(<field>)
Description: The options that you can specify to sort the events.

<field>
Syntax: <string>
Description: The name of the field to sort.

auto
Syntax: auto(<field>)
Description: Determine automatically how to sort the field values.

ip
Syntax: ip(<field>)
Description: Interpret the field values as IP addresses.

num
Syntax: num(<field>)
**Description:** Interpret the field values as numbers.

```
str
```

**Syntax:** `str(<field>)`

**Description:** Order the field values by using the lexicographic order.

**Usage**

The `dedup` command is a **streaming command** or a **dataset processing command**, depending on which arguments are specified with the command. See [Command types](#).

Avoid using the `dedup` command on the `_raw` field if you are searching over a large volume of data. If you search the `_raw` field, the text of every event in memory is retained which impacts your search performance. This is expected behavior. This behavior applies to any field with high cardinality and large size.

**Lexicographical order**

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.

- Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically as 10, 100, 70, 9.
- Uppercase letters are sorted before lowercase letters.
- Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

**Examples**

**Example 1:**

Remove duplicates of results with the same 'host' value.

```
... | dedup host
```

**Example 2:**

Remove duplicates of results with the same 'source' value and sort the events by the '_time' field in ascending order.
Example 3:
Remove duplicates of results with the same 'source' value and sort the events by the '_size' field in descending order.

... | dedup source sortby +_time

Example 4:
For events that have the same 'source' value, keep the first 3 that occur and remove all subsequent events.

... | dedup 3 source

Example 5:
For events that have the same 'source' AND 'host' values, keep the first 3 that occur and remove all subsequent events.

... | dedup 3 source host

See also
uniq

Answers
Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the dedup command.

delete

Description
Using the delete command marks all of the events returned by the search as deleted. Subsequent searches do not return the marked events. No user, not even a user with admin permissions, is able to view this data after deletion. The delete command does not reclaim disk space.
Removing data is irreversible. If you want to get your data back after the data is deleted, you must re-index the applicable data sources.

You cannot run the `delete` command in a real-time search to delete events as they arrive.

**Syntax**

delete

**Usage**

The `delete` command can be accessed only by a user with the "delete_by_keyword" capability. By default, only the "can_delete" role has the ability to delete events. No other role, including the admin role, has this ability. You should create a special userid that you log on with when you intend to delete indexed data.

To use the `delete` command, run a search that returns the events you want deleted. Make sure that the search returns ONLY the events that you want to delete, and no other events. After you confirm that the results contain the data that you want to delete, pipe the search to the `delete` command.

The delete operator triggers a roll of hot buckets to warm in the affected indexes.

The output of the `delete` command is a table of the quantity of events removed by the fields splunk_server (the name of the indexer or search head), and index, as well as a rollup record for each server by index "__ALL__". The quantity of deleted events is in the `deleted` field. An `errors` field is also emitted, which will normally be 0.

**Note:** The `delete` command does not work if your events contain a field named `index` aside from the default `index` field that is applied to all events. If your events do contain an additional `index` field, you can use `eval` before invoking `delete`, as in this example:

```bash
index=fbus_summary latest=1417356000 earliest=1417273200 | eval index = "fbus_summary" | delete
```

**Permanently removing data from an index**

The `delete` command does not remove the data from your disk space. You must use the `clean` command from the CLI to permanently remove the data. The `clean`
command removes all of the data in an index. You cannot select the specific data that you want to remove. See Remove indexes and indexed data in *Managing Indexers and Clusters of Indexers*.

**Examples**

**Delete events with Social Security numbers**

Delete the events from the `insecure` index that contain strings that look like Social Security numbers. Use the `regex` command to identify events that contain the strings that you want to match.

1. Run the following search to ensure that you are retrieving the correct data from the `insecure` index.

```shell
index=insecure | regex _raw = "\d{3}-\d{2}-\d{4}"
```

2. If necessary, adjust the search to retrieve the correct data. Then add the `delete` command to the end of the search to delete the events.

```shell
index=insecure | regex _raw = "\d{3}-\d{2}-\d{4}" | delete
```

**Delete events that contain a specific word**

Delete events from the `imap` index that contain the word `invalid`.

```shell
index=imap invalid | delete
```

**Remove the Search Tutorial events**

Remove all of the Splunk Search Tutorial events from your index.

1. Login as a user with the `admin` role.
2. Click *Settings, Access controls* and create a new user with the `can_delete` role.
3. Log out as admin and log back in as the user with the `can_delete` role.
4. Set the time range picker to *All time*.
5. Run the following search to retrieve all of the Search Tutorial events.

```shell
source=tutorialdata.zip:*
```

6. Confirm that the search is retrieving the correct data.
7. Add the `delete` command to the end of the search criteria and run the search again.

```shell
source=tutorialdata.zip:* | delete
```
The events are removed from the index.
8. Log out as the user with the can_delete role.

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the delete command.

**delta**

**Description**

Computes the difference between nearby results using the value of a specific numeric field. For each event where field is a number, the delta command computes the difference, in search order, between the field value for the event and the field value for the previous event. The delta command writes this difference into newfield.

If the newfield argument is not specified, then the delta command uses delta(field).

If field is not a number in either of the two values, no output field is generated.

**Note:** The delta command works on the events in the order they are returned by search. By default, the events for historical searches are in reverse time order from new events to old events. Values ascending over time show negative deltas. For real-time search, the events are compared in the order they are received. In the general case, the delta could be applied after any sequence of commands, so there is no input order guaranteed. For example, if you sort your results by an independent field and then use the delta command, the produced values are the deltas in that specific order.

**Syntax**

delta (field [AS newfield]) [p=int]

**Required arguments**

- field

Syntax: <field-name>
Description: The name of a field to analyze.
Optional arguments

newfield

Syntax: <string>
Description: Write output to this field.
Default: delta(field-name)

p

Syntax: p=<int>
Description: Specifies how many results prior to the current result to use for the comparison to the value in field in the current result. The prior results are determined by the search order, which is not necessarily chronological order. If p=1, compares the current result value against the value in the first result prior to the current result. If p=2, compares the current result value against the value in the result that is two results prior to the current result, and so on.
Default: 1

Basic examples

1. Calculate the difference in activity

With the logs from a cable TV provider, sourcetype=tv, you can analyze broadcasting ratings, customer preferences, and so on. Which channels do subscribers watch the most, activity=view, and how long do the subscribers stay on those channels?

sourcetype=tv activity="View" | sort - _time | delta _time AS timeDeltaS | eval timeDeltaS=abs(timeDeltaS) | stats sum(timeDeltaS) by ChannelName

2. Calculate the difference between that current value and the 3rd previous value

Compute the difference between current value of count and the 3rd previous value of count and store the result in the default field, delta(fieldname), which in this example is delta(count).

... | delta count p=3
3. Calculate the difference between that current value and the previous value and rename the result field

For each event where 'count' exists, compute the difference between count and its previous value and store the result in the field countdiff.

... | delta count AS countdiff

Extended examples

1. Calculate the difference in the number of purchases between the top 10 buyers

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

Find the top ten people who bought something yesterday, count how many purchases they made and the difference in the number of purchases between each buyer.

sourcetype=access_* status=200 action=purchase | top clientip | delta count p=1

- The purchase events, action=purchase, are piped into the top command to find the top ten users, based on clientip, who bought something.
- These results, which include a count for each clientip are then piped into the delta command to calculate the difference between the count value of one event and the count value of the event preceding it, using the p-1 argument.
- By default, this difference is saved in a new field called delta(count).
- The first event does not have a delta(count) value.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>clientip</th>
<th>count</th>
<th>percent</th>
<th>delta(count)</th>
</tr>
</thead>
<tbody>
<tr>
<td>87.194.216.51</td>
<td>134</td>
<td>2.565084</td>
<td></td>
</tr>
<tr>
<td>128.241.220.82</td>
<td>95</td>
<td>1.818530</td>
<td>-39</td>
</tr>
<tr>
<td>211.166.11.101</td>
<td>91</td>
<td>1.741960</td>
<td>-4</td>
</tr>
<tr>
<td>107.3.146.207</td>
<td>72</td>
<td>1.378254</td>
<td>-19</td>
</tr>
</tbody>
</table>
2. Calculate the difference in time between recent events

This example uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the USGS Earthquake Feeds and add it as an input.

Calculate the difference in time between each of the recent earthquakes in Alaska. Run the search using the time range All time.

```
source=all_month.csv place=*alaska* | delta _time p=1 | rename delta(_time) AS timeDeltaS | eval timeDeltaS=abs(timeDeltaS) | eval "Time Between Quakes"=tostring(timeDeltaS,"duration") | table place, _time, "Time Between Quakes"
```

- This example searches for earthquakes in Alaska.

The `delta` command is used to calculate the difference in the timestamps, `_time`, between each earthquake and the one immediately before it. By default the difference is placed in a new field called `delta(_time)`. The time is in seconds.

- The `rename` command is used to change the default name to `timeDeltaS`.
- An `eval` command is used with the `abs` function to convert the time into the absolute value of the time. This conversion is necessary because the differences between one earthquake and the earthquake immediately before it result in negative values.
- Another `eval` command is used with the `tostring` function to convert the time, in seconds, into a string value. The `duration` argument is part of the `tostring` function that specifies to convert the value to a readable time format HH:MM:SS.
The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>place</th>
<th>_time</th>
<th>Time Between Quakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>32km N of Anchor Point, Alaska</td>
<td>2018-04-04 19:51:19.147</td>
<td></td>
</tr>
<tr>
<td>34km NE of Valdez, Alaska</td>
<td>2018-04-04 16:21:57.040</td>
<td>00:04:17.701</td>
</tr>
<tr>
<td>23km NE of Fairbanks, Alaska</td>
<td>2018-04-04 16:10:05.595</td>
<td>00:11:51.445</td>
</tr>
<tr>
<td>53km SSE of Cantwell, Alaska</td>
<td>2018-04-04 16:07:04.498</td>
<td>00:03:01.097</td>
</tr>
<tr>
<td>114km NNE of Arctic Village, Alaska</td>
<td>2018-04-04 12:08:00.384</td>
<td>01:49:05.796</td>
</tr>
<tr>
<td>109km W of Cantwell, Alaska</td>
<td>2018-04-04 11:25:36.307</td>
<td>00:23:45.509</td>
</tr>
</tbody>
</table>

3. Calculate the difference in time between consecutive transactions

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

Calculate the difference in time between consecutive transactions.

```
sourcetype=access_* | transaction JSESSIONID clientip startswith="view" endswith="purchase" | delta _time AS timeDelta p=1 | eval timeDelta=abs(timeDelta) | eval timeDelta=tostring(timeDelta,"duration")
```
• This example groups events into transactions if they have the same values of JSESSIONID and clientip.
• The beginning of a transaction is defined by an event that contains the string view. The end of a transaction is defined by an event that contains the string purchase. The keywords view and purchase correspond to the values of the action field. You might also notice other values for the code>action</code> field, such as addtocart and remove.
• The transactions are then piped into the delta command, which uses the _time field to calculate the time between one transaction and the transaction immediately preceding it. Specifically the difference between the timestamp for the last event in the transaction and the timestamp in the last event in the previous transaction.
• The search renames the time change as timeDelta.
• An eval command is used with the abs function to convert the time into the absolute value of the time. This conversion is necessary because the differences between one transaction and the previous transaction result in negative values.
• Another eval command is used with the tostring function to convert the time, in seconds, into a string value. The duration argument is part of the tostring function that specifies to convert the value to a readable time format HH:MM:SS.

See also
accum, autoregress, streamstats, trendline

Answers
Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the delta command.
**diff**

**Description**

Compares two search results and returns the line-by-line difference, or comparison, of the two. The two search results compared are specified by the two position values `position1` and `position2`. These values default to 1 and 2 to compare the first two results.

By default, the text `_raw` field of the two search results is compared. Other fields can be compared by selecting another field using `attribute`.

**Syntax**

diff [position1=int] [position2=int] [attribute=string] [diffheader=bool] [context=bool] [maxlen=int]

**Optional arguments**

`position1`
- **Datatype:** `<int>`
- **Description:** Of the table of input search results, selects a specific search result to compare to position2.
- **Default:** `position1=1` and refers to the first search result.

`position2`
- **Datatype:** `<int>`
- **Description:** Of the table of input search results, selects a specific search result to compare to position1. This value must be greater than `position1`.
- **Default:** `position2=2` and refers to the second search result.

`attribute`
- **Datatype:** `<field>`
- **Description:** The field name to be compared between the two search results.
- **Default:** `attribute=_raw`, which refers to the text of the event or result.

`diffheader`
- **Datatype:** `<bool>`
- **Description:** If true, show the traditional diff header, naming the "files" compared. The diff header makes the output a valid diff as would be expected by the programmer command-line `patch` command.
Default: \texttt{diffheader=false}.

\textbf{context}

\textbf{Datatype:} <\texttt{bool}>
\textbf{Description:} If true, selects context-mode diff output as opposed to the default unified diff output.
\textbf{Default:} context=false, or unified.

\textbf{maxlen}

\textbf{Datatype:} <\texttt{int}>
\textbf{Description:} Controls the maximum content in bytes diffed from the two events. If maxlen=0, there is no limit.
\textbf{Default:} maxlen=100000, which is 100KB.

\section*{Examples}

\textbf{Example 1:}

Compare the "ip" values of the first and third search results.

\ldots | \texttt{diff pos1=1 pos2=3 attribute=ip}

\textbf{Example 2:}

Compare the 9th search results to the 10th.

\ldots | \texttt{diff position1=9 position2=10}

\section*{See also}

\texttt{set}

\section*{Answers}

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the \texttt{diff} command.

\texttt{erex}
Description

Use the `erex` command to extract data from a field when you do not know the regular expression to use. The command automatically extracts field values that are similar to the example values you specify.

If you specify a `field` argument, the values extracted from the `fromfield` argument are saved to the `field`. Otherwise, the search returns a regular expression that you can then use with the `rex` command to extract the field.

Syntax

`erex [<field>] examples=<string> [counterexamples=<string>] [fromfield=<field>] [maxtrainers=<int>]`

**Required arguments**

examples

**Syntax:** examples=<string>,<string>...

**Description:** A comma separated list of example values for the information to extract and save into a new field. Use quotation marks around the list if the list contains spaces. For example: "port 3351, port 3768".

**Optional arguments**

counterexamples

**Syntax:** counterexamples=<string>,<string>,...

**Description:** A comma-separated list of example values that represent information not to be extracted.

field

**Syntax:** <string>

**Description:** A name for a new field that will take the values extracted from `fromfield`. If `field` is not specified, values are not extracted, but the resulting regular expression is generated and placed as a message under the Jobs menu in Splunk Web. That regular expression can then be used with the `rex` command for more efficient extraction.

fromfield

**Syntax:** fromfield=<field>

**Description:** The name of the existing field to extract the information from and save into a new field.
Default: _raw

maxtrainers
Syntax: maxtrainers=<int>
Description: The maximum number values to learn from. Must be between 1 and 1000.
Default: 100

Usage

The values specified in the examples and counterexample arguments must exist in the events that are piped into the erex command. If the values do not exist, the command fails.

To make sure that the erex command works against your events, first run the search that returns the events you want without the erex command. Then copy the field values that you want to extract and use those for the example values with the erex command.

Examples

1. Extract values based on an example

The following search extracts out month and day values like 7/01 and puts the values into the monthday attribute.

   ... | erex monthday examples="7/01"

2. Extract values based on examples and counter examples

The following search extracts out month and day values like 7/01 and 7/02, but not patterns like 99/2. The extracted values are put into the monthday attribute.

   ... | erex monthday examples="7/01, 07/02" counterexamples="99/2"

3. Extract values based on examples and return the most common values

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

Determine which are the most common ports used by potential attackers.
1. Run a search to find examples of the port values, where there was a failed login attempt.

```
sourcetype=secure* port "failed password"
```

2. Then use the `erex` command to extract the port field. You must specify several examples with the `erex` command. Use the `top` command to return the most common port values. By default the `top` command returns the top 10 values.

```
sourcetype=secure* port "failed password" | erex port examples="port 3351, port 3768" | top port
```

This search returns a table with the count of top ports that match the search.

The results appear on the Statistics tab and look something like this:

| port   | count | percent 
|--------|-------|----------
| port 2444 | 20    | 0.060145 |
| port 3281 | 19    | 0.057138 |
| port 2842 | 19    | 0.057138 |
| port 2760 | 19    | 0.057138 |
| port 1174 | 19    | 0.057138 |
| port 4955 | 18    | 0.054130 |
| port 1613 | 18    | 0.054130 |
| port 1059 | 18    | 0.054130 |
| port 4542 | 17    | 0.051123 |
| port 4519 | 17    | 0.051123 |

3. Click the Jobs menu to see the generated regular expression based on your examples. You can use the `rex` command with the regular expression.
instead of using the `erex` command. The regular expression for this search example is:
```bash
| rex (?i)^{?:[^.]*.}{3}\d+\s+(?P<port>\w+\s+\d+) for this search example.
```

You can replace the `erex` command with the `rex` command and generated regular expression in your search. For example:
```
sourcetype=secure* port "failed password" | rex
  (?i)^{?:[^.]*.}{3}\d+\s+(?P<port>\w+\s+\d+) | top port
```

Using the `rex` command with a regular expression is more cost effective than using the `erex` command.

**See also**

`extract`, `kvform`, `multikv`, `regex`, `rex`, `xmlkv`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `erex` command.

**eval**

**Description**

The `eval` command calculates an expression and puts the resulting value into a search results field.

- If the field name that you specify does not match a field in the output, a new field is added to the search results.
• If the field name that you specify matches a field name that already exists in the search results, the results of the eval expression overwrite the values in that field.

The eval command evaluates mathematical, string, and boolean expressions.

You can chain multiple eval expressions in one search using a comma to separate subsequent expressions. The search processes multiple eval expressions left-to-right and lets you reference previously evaluated fields in subsequent expressions.

**Difference between eval and stats commands**

The stats command calculates statistics based on fields in your events. The eval command creates new fields in your events by using existing fields and an arbitrary expression.

![Image showing before and after eval command](image)

**Syntax**

```
eval <field>=<expression>["", <field>=<expression>]...
```

**Required arguments**

- **field**
  
  **Syntax:** `<string>`
  
  **Description:** A destination field name for the resulting calculated value. If the field name already exists in your events, eval overwrites the value.

- **expression**
  
  **Syntax:** `<string>`
  
  **Description:** A combination of values, variables, operators, and functions that will be executed to determine the value to place in your destination field.

The syntax of the eval expression is checked before running the search, and an exception is thrown for an invalid expression.
* The result of an eval expression cannot be a Boolean. 
* If, at search time, the expression cannot be evaluated successfully for a given event, the eval command erases the resulting field.
* If the expression references a field name that contains non-alphanumeric characters, other than the underscore ( _ ) character, the field name needs to be surrounded by **single quotation marks**. For example, if the field name is server-1 you specify the field name like this new=count+'server-1'.
* If the expression references a literal string, that string needs to be surrounded by **double quotation marks**. For example, if the string you want to use is server- you specify the string like this new="server-".host.

**Usage**

The eval command is a distributable streaming command. See Command types.

**General**

You must specify a field name for the results that are returned from your eval command expression. You can specify a name for a new field or for an existing field.

If the field name that you specify matches an existing field name, the values in the existing field are replaced by the results of the eval expression.

Numbers and strings can be assigned to fields, while booleans cannot be assigned. However you can convert booleans and nulls to strings using the tostring() function, which can be assigned to fields.

During calculations, numbers are double precision floating point numbers subject to all the usual behaviors of floating point numbers. Operations resulting in NaN assigned to a field will result in "nan". Positive and negative overflow will result in "inf" and "-inf". Division by zero will result in a null field.

If you are using a search as an argument to the eval command and functions, you cannot use a saved search name; you must pass a literal search string or a field that contains a literal search string (like the 'search' field extracted from index=_audit events).
## Functions

You can use a wide range of functions with the `eval` command. For general information about using functions, see [Evaluation functions](#).

The following table lists the supported functions by type of function. Use the links in the table to learn more about each function, and to see examples.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Comparison and Conditional functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>case(X,&quot;Y&quot;,...)</td>
<td>in(VALUE-LIST)</td>
<td>nullif(X,Y)</td>
</tr>
<tr>
<td>cidrmatch(&quot;X&quot;,Y)</td>
<td>like(TEXT, PATTERN)</td>
<td>searchmatch(X,Y)</td>
</tr>
<tr>
<td>coalesce(X,...)</td>
<td>match(SUBJECT, &quot;REGEX&quot;)</td>
<td>validate(X)</td>
</tr>
<tr>
<td>false()</td>
<td>null()</td>
<td></td>
</tr>
<tr>
<td>if(X,Y,Z)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Conversion functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>printf(&quot;format&quot;,arguments)</td>
<td>tonumber(NUMSTR,BASE)</td>
<td>tostring(X,Y)</td>
</tr>
<tr>
<td><strong>Cryptographic functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>md5(X)</td>
<td>sha256(X)</td>
<td>sha512(X)</td>
</tr>
<tr>
<td>shal(X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Date and Time functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>now()</td>
<td>strftime(X,Y)</td>
<td>time()</td>
</tr>
<tr>
<td>relative_time(X,Y)</td>
<td>strptime(X,Y)</td>
<td></td>
</tr>
<tr>
<td><strong>Informational functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isbool(X)</td>
<td>isnull(X)</td>
<td>isstr(X)</td>
</tr>
<tr>
<td>isint(X)</td>
<td>isnull(X)</td>
<td>isnull(X)</td>
</tr>
<tr>
<td>isnotnull(X)</td>
<td>isnum(X)</td>
<td>isnull(X)</td>
</tr>
<tr>
<td><strong>Mathematical functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abs(X)</td>
<td>floor(X)</td>
<td>pow(X,Y)</td>
</tr>
<tr>
<td>ceiling(X)</td>
<td>ln(X)</td>
<td>round(X,Y)</td>
</tr>
<tr>
<td>exact(X)</td>
<td>log(X,Y)</td>
<td>sigfig(X)</td>
</tr>
<tr>
<td>exp(X)</td>
<td>pi()</td>
<td>sqrt(X)</td>
</tr>
<tr>
<td><strong>Multivalue eval functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commands(X)</td>
<td>mvfilter(X)</td>
<td>mvrange(X,Y,Z)</td>
</tr>
<tr>
<td>mvappend(X,...)</td>
<td>mvfind(MVFIELD,&quot;REGEX&quot;)</td>
<td>mvsort(X,Y)</td>
</tr>
<tr>
<td>mvcount(MVFIELD)</td>
<td>mvindex(MVFIELD,STARTINDEX,ENDINDEX)</td>
<td>mvzip(X,Y,Z)</td>
</tr>
<tr>
<td>mvdedup(X)</td>
<td>mvjoin(MVFIELD,STR)</td>
<td>split(X)</td>
</tr>
<tr>
<td><strong>max(X,...)</strong></td>
<td>min(X,...)</td>
<td>random()</td>
</tr>
<tr>
<td>Type of function</td>
<td>Supported functions and syntax</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------</td>
<td></td>
</tr>
<tr>
<td>Statistical eval functions</td>
<td>len(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lower(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ltrim(X,Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>replace(X,Y,Z)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rtrim(X,Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>spath(X,Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>substr(X,Y,Z)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>trim(X,Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>upper(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>urldecode(X)</td>
<td></td>
</tr>
<tr>
<td>Text functions</td>
<td>acos(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>acosh(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>asin(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>asinh(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>atan(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>atan2(X,Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>atanh(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cos(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cosh(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hypot(X,Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sin(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sinh(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tan(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tanh(X)</td>
<td></td>
</tr>
<tr>
<td>Trigonometry and Hyperbolic functions</td>
<td>atan2(X,Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cos(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cosh(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hypot(X,Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sin(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sinh(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tan(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tanh(X)</td>
<td></td>
</tr>
</tbody>
</table>

**Operators**

The following table lists the basic operations you can perform with the `eval` command. For these evaluations to work, the values need to be valid for the type of operation. For example, with the exception of addition, arithmetic operations might not produce valid results if the values are not numerical. When concatenating values, Splunk software reads the values as strings, regardless of the value.

<table>
<thead>
<tr>
<th>Type</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>+ - * / %</td>
</tr>
<tr>
<td>Concatenation</td>
<td>.</td>
</tr>
<tr>
<td>Boolean</td>
<td>AND OR NOT XOR &lt; &gt; &lt;= &gt;= != = == LIKE</td>
</tr>
</tbody>
</table>

**Operators that produce numbers**

- The plus (+) operator accepts two numbers for addition, or two strings for concatenation.
- The subtraction (-), multiplication (*), division (/), and modulus (%) operators accept two numbers.

**Operators that produce strings**

- The period (.) operator concatenates both strings and number. Numbers are concatenated in their string represented form.
Operators that produce booleans

- The AND, OR, and XOR operators accept two Boolean values.
- The <>, <=, !=, and == operators accept two numbers or two strings. The != and == operators accept two numbers or two strings. The single equal sign (=) is a synonym for the double equal sign (==).
- The LIKE operator accepts two strings. This is a pattern match similar to what is used in SQL. For example, string LIKE pattern. The pattern operator supports literal text, a percent (%) character for a wildcard, and an underscore (_) character for a single character match. For example, field LIKE "a%b_" matches any string starting with a, followed by anything, followed by b, followed by one character.

Field names

To specify a field name with multiple words, you can either concatenate the words, or use single quotation marks when you specify the name. For example, to specify the field name Account ID you can specify AccountID or 'Account ID'.

To specify a field name with special characters, such as a period, use single quotation marks. For example, to specify the field name Last.Name use 'Last.Name'.

You can use the value of another field as the name of the destination field by using curly brackets, { }. For example, if you have an event with the following fields, aName=counter and aValue=1234. Use | eval {aName}=aValue to return counter=1234.

Calculated fields

You can use eval statements to define calculated fields by defining the eval statement in props.conf. If you are using Splunk Cloud, you can define calculated fields using Splunk Web, by choosing Settings > Fields > Calculated Fields. When you run a search, Splunk software evaluates the statements and creates fields in a manner similar to that of search time field extraction. Setting up calculated fields means that you no longer need to define the eval statement in a search string. Instead, you can search on the resulting calculated field directly.

You can use calculated fields to move your commonly used eval statements out of your search string and into props.conf, where they will be processed behind the scenes at search time. With calculated fields, you can change the search
from:

sourcetype="cisco_esa" mailfrom=* | eval
accountname=split(mailfrom,"@"), from_user=mvindex(accountname,0),
from_domain=mvindex(accountname,-1) | table mailfrom, from_user,
from_domain

to this search:

sourcetype="cisco_esa" mailfrom=* | table mailfrom, from_user,
from_domain

In this example, the three eval statements that were in the search--that defined
the accountname, from_user, and from_domain fields--are now computed behind
the scenes when the search is run for any event that contains the extracted field
mailfrom field. You can also search on those fields independently once they’re
set up as calculated fields in props.conf. You could search on
from_domain=email.com, for example.

For more information about calculated fields, see About calculated fields in the
Knowledge Manager Manual.

**Search event tokens**

If you are using the `eval` command in search event tokens, some of the
evaluation functions might be unavailable or have a different behavior. See
Custom logic for search tokens in *Dashboards and Visualizations* for information
about the evaluation functions that you can use with search event tokens.

**Basic Examples**

1. **Create a new field that contains the result of a calculation**

Create a new field called `velocity` in each event. Calculate the velocity by
dividing the values in the distance field by the values in the time field.

```plaintext
... | eval velocity=distance/time
```

2. **Use the `if` function to analyze field values**

Create a field called `error` in each event. Using the `if` function, set the value in
the error field to OK if the status value is 200. Otherwise set the error field value
to Problem.

```plaintext
... | eval error=if(status=200, "OK", "Problem")
```
3. Convert values to lowercase

Create a new field in each event called low-user. Using the lower function, populate the field with the lowercase version of the values in the username field. Because the field name contains a dash (-), the name must be enclosed in single quotation marks.

```
... | eval low-user = lower(username)
```

4. Use the value of one field as the name for a new field

In this example, use each value of the field counter to make a new field name. Assign to the new field the value of the Value field. See Field names under the Usage section.

```
index=perfmon sourcetype=Perfmon* counter=* Value=* | eval {counter} = Value
```

5. Set sum_of_areas to be the sum of the areas of two circles

```
... | eval sum_of_areas = pi() * pow(radius_a, 2) + pi() * pow(radius_b, 2)
```

6. Set status to some simple http error codes

```
... | eval error_msg = case(error == 404, "Not found", error == 500, "Internal Server Error", error == 200, "OK")
```

7. Concatenate values from two fields

Use the period (.) character to concatenate the values in first_name field with the values in the last_name field. Quotation marks are used to insert a space character between the two names. When concatenating, the values are read as strings, regardless of the actual value.

```
... | eval full_name = first_name." ".last_name
```

8. Separate multiple eval operations with a comma

You can specify multiple eval operations by using a comma to separate the operations. In the following search the full_name evaluation uses the period (.) character to concatenate the values in the first_name field with the values in the
last_name field. The low_name evaluation uses the lower function to convert the full_name evaluation into lowercase.

... | eval full_name = first_name." ".last_name, low_name = lower(full_name)

9. Display timechart of the avg of cpu_seconds by processor, rounded to 2 decimals

... | timechart eval(round(avg(cpu_seconds),2)) by processor

10. Convert a numeric field value to a string with commas and 2 decimals

If the original value of x is 1000000, this returns x as 1,000,000.

... | eval x=tostring(x,"commas")

To include a currency symbol at the beginning of the string:

... | eval x="\$.tostring(x,"commas")

This returns x as $1,000,000.

Extended Examples

11. Coalesce a field from two different source types, create a transaction of events

This example shows how you might coalesce a field from two different source types and use that to create a transaction of events. sourcetype=A has a field called number, and sourcetype=B has the same information in a field called subscriberNumber.

sourcetype=A OR sourcetype=B | eval phone=coalesce(number,subscriberNumber) | transaction phone maxspan=2m

The eval command is used to add a common field, called phone, to each of the events whether they are from sourcetype=A or sourcetype=B. The value of phone is defined, using the coalesce() function, as the values of number and subscriberNumber. The coalesce() function takes the value of the first non-NULL field (that means, it exists in the event).

Now, you're able to group events from either source type A or B if they share the same phone value.
12. Separate events into categories, count and display minimum and maximum values

This example uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), and so forth, for each earthquake recorded.

You can download a current CSV file from the USGS Earthquake Feeds and upload the file to your Splunk instance if you want follow along with this example.

Earthquakes occurring at a depth of less than 70 km are classified as shallow-focus earthquakes, while those with a focal-depth between 70 and 300 km are commonly termed mid-focus earthquakes. In subduction zones, deep-focus earthquakes may occur at much greater depths (ranging from 300 up to 700 kilometers).

To classify recent earthquakes based on their depth, you use the following search.

```
source=all_month.csv | eval Description=case(depth<=70, "Shallow", depth>70 AND depth<=300, "Mid", depth>300, "Deep") | stats count min(mag) max(mag) by Description
```

The `eval` command is used to create a field called `Description`, which takes the value of "Shallow", "Mid", or "Deep" based on the Depth of the earthquake. The `case()` function is used to specify which ranges of the depth fits each description. For example, if the depth is less than 70 km, the earthquake is characterized as a shallow-focus quake; and the resulting `Description` is Shallow.

The search also pipes the results of the `eval` command into the `stats` command to count the number of earthquakes and display the minimum and maximum magnitudes for each Description.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>Description</th>
<th>count</th>
<th>min(Mag)</th>
<th>max(Mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep</td>
<td>35</td>
<td>4.1</td>
<td>6.7</td>
</tr>
<tr>
<td>Mid</td>
<td>635</td>
<td>0.8</td>
<td>6.3</td>
</tr>
<tr>
<td>Shallow</td>
<td>6236</td>
<td>-0.60</td>
<td>7.70</td>
</tr>
</tbody>
</table>
13. Find IP addresses and categorize by network using eval functions cidrmatch and if

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range *Yesterday* when you run the search.

In this search, you’re finding IP addresses and classifying the network they belong to.

```
sourcetype=access_* | eval network=if(cidrmatch("182.236.164.11/16", clientip), "local", "other")
```

This example uses the `cidrmatch()` function to compare the IP addresses in the `clientip` field to a subnet range. The search also uses the `if()` function, which says that if the value of `clientip` falls in the subnet range, then the `network` field value is `local`. Otherwise, `network=other`.

The `eval` command does not do any special formatting to your results. The command creates a new field based on the eval expression you specify.

In the fields sidebar, click on the `network` field. In the popup, next to *Selected* click *Yes* and close the popup. Now you can see, inline with your search results, which IP addresses are part of your `local` network and which are not. Your events list looks something like this:
Another option for formatting your results is to pipe the results of `eval` to the `table` command to display only the fields of interest to you.

**Note:** This example just illustrates how to use the `cidrmatch` function. If you want to classify your events and quickly search for those events, the better approach is to use event types. Read more about event types in the *Knowledge manager manual*.

### 14. Extract information from an event into a separate field, create a multivalue field

This example uses sample email data. You should be able to run this search on any email data by replacing the `sourcetype=cisco:esa` with the `sourcetype` value and the `mailfrom` field with email address field name in your data. For example, the email might be `To`, `From`, or `Cc`.

Use the email address field to extract the name and domain. The `eval` command in this search contains multiple expressions, separated by commas.

```
sourcetype="cisco:esa" mailfrom=* | eval accountname=split(mailfrom,"@"), from_user=mvindex(accountname,0), from_domain=mvindex(accountname,-1) | table mailfrom, from_user, from_domain
```

- The `split()` function is used to break the `mailfrom` field into a multivalue field called `accountname`. The first value of `accountname` is everything before the "@" symbol, and the second value is everything after.
- The `mvindex()` function is used to set `from_user` to the first value in `accountname` and to set `from_domain` to the second value in `accountname`.
- The results of the `eval` expressions are then piped into the `table` command.

You can see the the original `mailfrom` values and the new `from_user` and `from_domain` values in the results table. The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>mailfrom</th>
<th>from_user</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:na.lui@sample.net">na.lui@sample.net</a></td>
<td>na.lui</td>
</tr>
<tr>
<td><a href="mailto:MAILER-DAEMON@hcp2mailsec.sample.net">MAILER-DAEMON@hcp2mailsec.sample.net</a></td>
<td>MAILER-DAEMON</td>
</tr>
<tr>
<td>M&amp;<a href="mailto:MService@example.com">MService@example.com</a></td>
<td>M&amp;MService</td>
</tr>
<tr>
<td><a href="mailto:AlexMartin@oursample.de">AlexMartin@oursample.de</a></td>
<td>AlexMartin</td>
</tr>
<tr>
<td><a href="mailto:Exit_Desk@sample.net">Exit_Desk@sample.net</a></td>
<td>Exit_Desk</td>
</tr>
</tbody>
</table>

324
Note: This example was written to demonstrate how to use an eval function to identify the individual values of a multivalue fields. Because this particular set of email data did not have any multivalue fields, the example creates a multivalue field, accountname, from a single value field, mailfrom.

15. Categorize events using the match function

This example uses sample email data. You should be able to run this search on any email data by replacing the sourcetype=cisco:esa with the sourcetype value and the mailfrom field with email address field name in your data. For example, the email might be To, From, or Cc).

This example classifies where an email came from based on the email address domain. The .com, .net, and .org addresses are considered local, while anything else is considered abroad. There are many domain names. Of course, domains that are not .com, .net, or .org are not necessarily from abroad. This is just an example.

The eval command in this search contains multiple expressions, separated by commas.

sourcetype="cisco:esa" mailfrom=* | eval accountname=split(mailfrom,"@"), from_domain=mvindex(accountname,-1), location=if(match(from_domain, "[\^\n\r\s]+.(com|net|org)"), "local", "abroad") | stats count BY location

The first half of this search is similar to previous example. The split() function is used to break up the email address in the mailfrom field. The mvindex function defines the from_domain as the portion of the mailfrom field after the @ symbol.

Then, the if() and match() functions are used.

- If the from_domain value ends with a .com, .net., or .org, the location field is assigned the value local.
- If from_domain does not match, location is assigned the value abroad.

The eval results are then piped into the stats command to count the number of results for each location value.
The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>location</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>abroad</td>
<td>3543</td>
</tr>
<tr>
<td>local</td>
<td>14136</td>
</tr>
</tbody>
</table>

Note: This example merely illustrates using the match() function. If you want to classify your events and quickly search for those events, the better approach is to use event types. Read more about event types in the Knowledge manager manual.

16. Convert the duration of transactions into more readable string formats

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

When you use the transaction command, as shown in the following search, it calculates the length of time for the transaction. A new field, called duration, is automatically added to the results. The duration is the time between the first and last events in the transaction.

```
sourcetype=access_* | transaction clientip maxspan=10m
```

In the Interesting fields list, click on the duration field to see the top 10 values for duration. The values are displayed in seconds. Click Yes to add the field to the Selected fields list.

You can use the eval command to reformat a numeric field into a more readable string format. The following search uses the tostring() function with the "duration" option to convert the values in the duration field into a string formatted as HH:MM:SS.

```
sourcetype=access_* | transaction clientip maxspan=10m | eval durationstr=tostring(duration,"duration")
```

The search defines a new field, durationstr, for the reformatted duration values. In the Interesting fields list, click on the durationstr field and select Yes to add the field to the Selected fields list. The values for the fields now appear in the set of fields below each transaction. The following image shows how your search results should look:
See also

Functions
  Evaluation functions

Commands
  where

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the eval command.

eventcount

Description

Returns the number of events in the specified indexes.

Syntax

| eventcount [index=<string>]... [summarize=<bool>] [report_size=<bool>] [list_vix=<bool>] |

Required arguments

None.
Optional arguments

index
Syntax: index=<string>
Description: A name of the index report on, or a wildcard matching many
indexes to report on. You can specify this argument multiple times, for
example index=* index=_*.
Default: If no index is specified, the command returns information about
the default index.

list_vix
Syntax: list_vix=<bool>
Description: Specify whether or not to list virtual indexes. If
list_vix=false, the command does not list virtual indexes.
Default: true

report_size
Syntax: report_size=<bool>
Description: Specify whether or not to report the index size. If
report_size=true, the command returns the index size in bytes.
Default: false

summarize
Syntax: summarize=<bool>
Description: Specifies whether or not to summarize events across all
peers and indexes. If summarize=false, the command splits the event
counts by index and search peer.
Default: true

Usage

The eventcount command is a report-generating command. See Command
types.

Generating commands use a leading pipe character and should be the first
command in a search.

Specifying a time range has no effect on the results returned by the eventcount
command. All of the events on the indexes you specify are counted.
**Specifying indexes**

You cannot specify indexes to exclude from the results. For example, `index!=foo` is not valid syntax.

You can specify the `index` argument multiple times. For example:

```
|eventcount summarize=false index=_audit index=main
```

**Examples**

**Example 1:**

Display a count of the events in the default indexes from all of the search peers. A single count is returned.

```
| eventcount
```

**Example 2:**

Return the number of events in only the internal default indexes. Include the index size, in bytes, in the results.

```
| eventcount summarize=false index=_* report_size=true
```

The results appear on the Statistics tab and should be similar to the results shown in the following table.

<table>
<thead>
<tr>
<th>count</th>
<th>index</th>
<th>server</th>
<th>size_bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>52550</td>
<td>_audit</td>
<td>buttercup-mbpr15.sv.splunk.com</td>
<td>7217152</td>
</tr>
<tr>
<td>1423010</td>
<td>_internal</td>
<td>buttercup-mbpr15.sv.splunk.com</td>
<td>122138624</td>
</tr>
<tr>
<td>22626</td>
<td>_introspection</td>
<td>buttercup-mbpr15.sv.splunk.com</td>
<td>98619392</td>
</tr>
<tr>
<td>10</td>
<td>_telemetry</td>
<td>buttercup-mbpr15.sv.splunk.com</td>
<td>135168</td>
</tr>
<tr>
<td>0</td>
<td>_thefishbucket</td>
<td>buttercup-mbpr15.sv.splunk.com</td>
<td>0</td>
</tr>
</tbody>
</table>

When you specify `summarize=false`, the command returns three fields: `count`, `index`, and `server`. When you specify `report_size=true`, the command returns the `size_bytes` field. The values in the `size_bytes` field are not the same as the index size on disk.
Example 3:

Return the event count for each index and server pair. Only the external indexes are returned.

| eventcount summarize=false index=* |

To return the count all of the indexes including the internal indexes, you must specify the internal indexes separately from the external indexes:

| eventcount summarize=false index=* index=_* |

See also

metadata, fieldsummary

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the eventcount command.

eventstats

Description

Adds summary statistics to all search results.

Generate summary statistics of all existing fields in your search results and saves those statistics in to new fields. The eventstats command is similar to the stats command. The difference is that with the eventstats command aggregation results are added inline to each event and added only if the aggregation is pertinent to that event.
Syntax

eventstats [allnum=<bool>] <stats-agg-term>... [by-clause]

Required arguments

<stats-agg-term>
   Syntax: <stats-func>(<evaled-field> | <wc-field>) [AS <wc-field>]
   Description: A statistical aggregation function. See Stats function options.
               The function can be applied to an eval expression, or to a field or set of
               fields. Use the AS clause to place the result into a new field with a name
               that you specify. You can use wild card characters in field names.

Optional arguments

 allnum
   Syntax: allnum=<bool>
   Description: If true, computes numerical statistics on each field if and
                only if all of the values of that field are numerical.
                Default: false

<by-clause>
   Syntax: BY <field-list>
   Description: The name of one or more fields to group by.

Stats function options

stats-func
   Syntax: The syntax depends on the function that you use. Refer to the
   table below.
   Description: Statistical and charting functions that you can use with the
               eventstats command. Each time you invoke the eventstats command,
               you can use one or more functions. However, you can only use one BY
               clause. See Usage.

   The following table lists the supported functions by type of function. Use
   the links in the table to see descriptions and examples for each function.
   For an overview about using functions with commands, see Statistical and
   charting functions.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and</th>
</tr>
</thead>
</table>
## Usage

In the limits.conf file, the `max_mem_usage_mb` setting in the `[default]` stanza is used to limit how much memory the `stats` and `eventstats` commands use to keep track of information. If the `eventstats` command reaches this limit, the command stops adding the requested fields to the search results. You can increase the limit, contingent on the available system memory.

Additionally, the `maxresultrows` setting in the `[searchresults]` stanza specifies the maximum number of results to return. The default value is 50,000. Increasing this limit can result in more memory usage.

Only users with file system access, such as system administrators, can edit the configuration files. Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

See How to edit a configuration file.

If you are using Splunk Cloud and want to change either of these settings, file a Support ticket.

### Functions and memory usage

Some functions are inherently more expensive, from a memory standpoint, than other functions. For example, the `distinct_count` function requires far more

<table>
<thead>
<tr>
<th>syntax</th>
<th>Aggregated functions</th>
<th>Event order functions</th>
<th>Multivalue stats and chart functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>avg()</code></td>
<td><code>count()</code></td>
<td><code>earliest()</code></td>
<td><code>list(X)</code></td>
</tr>
<tr>
<td><code>distinct_count()</code></td>
<td><code>estdc()</code></td>
<td><code>first()</code></td>
<td><code>values(X)</code></td>
</tr>
<tr>
<td><code>estdc_error()</code></td>
<td><code>max()</code></td>
<td><code>last()</code></td>
<td></td>
</tr>
<tr>
<td><code>count()</code></td>
<td><code>median()</code></td>
<td><code>range()</code></td>
<td></td>
</tr>
<tr>
<td><code>distinct_count()</code></td>
<td><code>min()</code></td>
<td><code>stdev()</code></td>
<td></td>
</tr>
<tr>
<td><code>estdc()</code></td>
<td><code>mode()</code></td>
<td><code>stdevp()</code></td>
<td></td>
</tr>
<tr>
<td><code>estdc_error()</code></td>
<td></td>
<td><code>sum()</code></td>
<td></td>
</tr>
<tr>
<td><code>exactperc&lt;int&gt;()</code></td>
<td></td>
<td><code>sumsq()</code></td>
<td></td>
</tr>
<tr>
<td><code>perc&lt;int&gt;()</code></td>
<td></td>
<td><code>upperperc&lt;int&gt;()</code></td>
<td></td>
</tr>
<tr>
<td><code>range()</code></td>
<td></td>
<td><code>var()</code></td>
<td></td>
</tr>
<tr>
<td><code>max()</code></td>
<td></td>
<td><code>varp()</code></td>
<td></td>
</tr>
<tr>
<td><code>median()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>min()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>mode()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stdev()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stdevp()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sum()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sumsq()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>upperperc&lt;int&gt;()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
memory than the count function. The values and list functions also can consume a lot of memory.

If you are using the distinct_count function without a split-by field or with a low-cardinality split-by by field, consider replacing the distinct_count function with the the estdc function (estimated distinct count). The estdc function might result in significantly lower memory usage and run times.

**Event order functions**

Using the first and last functions when searching based on time does not produce accurate results.

- To locate the first value based on time order, use the earliest function, instead of the first function.
- To locate the last value based on time order, use the latest function, instead of the last function.

For example, consider the following search.

```bash
index=test sourcetype=testDb | eventstats first(LastPass) as LastPass, last(_time) as mostRecentTestTime BY testCaseId | where startTime==LastPass OR _time==mostRecentTestTime | stats first(startTime) AS startTime, first(status) AS status, first(histID) AS currentHistId, last(histID) AS lastPassHistId BY testCaseId
```

When you use the stats and eventstats commands for ordering events based on time, use the earliest and latest functions.

The following search is the same as the previous search except the first and last functions are replaced with the earliest and latest functions.

```bash
index=test sourcetype=testDb | eventstats latest(LastPass) AS LastPass, earliest(_time) AS mostRecentTestTime BY testCaseId | where startTime==LastPass OR _time==mostRecentTestTime | stats latest(startTime) AS startTime, latest(status) AS status, latest(histID) AS currentHistId, earliest(histID) AS lastPassHistId BY testCaseId
```

**Examples**

**Example 1:** Compute the overall average duration and add 'avgdur' as a new field to each event where the 'duration' field exists

```bash
... | eventstats avg(duration) AS avgdur
```
**Example 2:** Same as Example 1 except that averages are calculated for each distinct value of date_hour and then each event gets the average for its particular value of date_hour.

```bash
... | eventstats avg(duration) AS avgdur BY date_hour
```

**Example 3:** This searches for spikes in error volume. You can use this search to trigger an alert if the count of errors is higher than average, for example.

```bash
eventtype="error" | eventstats avg(foo) AS avg | where foo>avg
```

**See also**

Commands
- stats, streamstats

Blogs
- Getting started with stats, eventstats and streamstats

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the eventstats command.

**extract**

**Description**

Extracts field-value pairs from the search results. The extract command works only on the _raw field. If you want to extract from another field, you must perform some field renaming before you run the extract command.

**Syntax**

```bash
extract [<extract-options>...] [<extractor-name>...]
```

**Required arguments**

None.
Optional arguments

<extract-options>
  Syntax: clean_keys=<bool> | kvdelim=<string> | limit=<int> | maxchars=<int> | mv_add=<bool> | pairdelim=<string> | reload=<bool> | segment=<bool>
  Description: Options for defining the extraction. See the Extract_options section in this topic.

<extractor-name>
  Syntax: <string>
  Description: A stanza in the transforms.conf file. This is used when the props.conf file does not explicitly cause an extraction for this source, sourcetype, or host.

Extract options

clean_keys
  Syntax: clean_keys=<bool>
  Description: Specifies whether to clean keys. Overrides CLEAN_KEYS in the transforms.conf file.
  Default: The value specified in the CLEAN_KEYS in the transforms.conf file.

kvdelim
  Syntax: kvdelim=<string>
  Description: A list of character delimiters that separate the key from the value.

limit
  Syntax: limit=<int>
  Description: Specifies how many automatic key-value pairs to extract.
  Default: 50

maxchars
  Syntax: maxchars=<int>
  Description: Specifies how many characters to look into the event.
  Default: 10240

mv_add
  Syntax: mv_add=<bool>
**Description:** Specifies whether to create multivalued fields. Overrides the value for the MV_ADD parameter in the
`transforms.conf` file.

**Default:** false

**pairdelim**

**Syntax:** pair=<string>
**Description:** A list of character delimiters that separate the key-value pairs from each other.

**reload**

**Syntax:** reload=<bool>
**Description:** Specifies whether to force reloading of the `props.conf` and `transforms.conf` files.

**Default:** false

**segment**

**Syntax:** segment=<bool>
**Description:** Specifies whether to note the locations of the key-value pairs with the results.

**Default:** false

**Usage**

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

**Alias**

The alias for the `extract` command is `kv`.

**Examples**

1. **Specify the delimiters to use for the field and value extractions**

Extract field-value pairs that are delimited by the pipe or semicolon characters (`|;`). Extract values of the fields that are delimited by the equal or colon characters (`=:`). The delimiters are individual characters. In this example the `="` or `:" character is used to delimit the key value. Similarly, a `|"` or `;" is used to delimit the field-value pair itself.

```
... | extract pairdelim="|;", kvdelim="=:
```
2. Extract field-value pairs and reload the field extraction settings

Extract field-value pairs and reload field extraction settings from disk.

... | extract reload=true

3. Rename a field to _raw to extract from that field

Rename the _raw field to a temporary name. Rename the field you want to extract from, to _raw. In this example the field name is uri_query.

... | rename _raw AS temp uri_query AS _raw | extract pairdelim="?&" kvdelim="=" | rename _raw AS uri_query temp AS _raw

4. Extract field-value pairs from a stanza in the transforms.conf file

Extract field-value pairs that are defined in the stanza 'access-extractions' in the transforms.conf file.

... | extract access-extractions

See also

tkform, multikv, rex, spath, xmlkv, xpath

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the extract command.

fieldformat

Description

With the fieldformat command you can use an <eval-expression> to change the format of a field value when the results render. This command changes the appearance of the results without changing the underlying value of the field.

Because commands that come later in the search pipeline cannot modify the formatted results, use the fieldformat command as late in the search pipeline as possible.
The fieldformat command does not apply to commands that export data, such as the outputcsv and outputlookup commands. The export retains the original data format and not the rendered format. If you want the format to apply to exported data, use the eval command instead of the fieldformat command.

**Syntax**

fieldformat <field>=<eval-expression>

**Required arguments**

<field>

**Description:** The name of a new or existing field, non-wildcarded, for the output of the eval expression.

<eval-expression>

**Syntax:** <string>

**Description:** A combination of values, variables, operators, and functions that represent the value of your destination field. You can specify only one <eval-expression> with the fieldformat command. To specify multiple formats you must use multiple fieldformat commands. See Examples.

For more information, see the eval command. For information about supported functions, see Usage.

**Usage**

The fieldformat command is a distributable streaming command. See Command types.

Time format variables are frequently used with the fieldformat command. See Date and time format variables.

**Functions**

You can use a wide range of functions with the fieldformat command. For general information about using functions, see Evaluation functions.

The following table lists the supported functions by type of function. Use the links in the table to learn more about each function, and to see examples.
<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Comparison and Conditional functions</strong></td>
<td>case(X,&quot;Y&quot;,...) [in(VALUE-LIST)] caseidrmatch(&quot;X&quot;,&quot;Y&quot;) coalesce(X,...)[false() if(X,Y,Z)]</td>
</tr>
<tr>
<td><strong>Conversion functions</strong></td>
<td>printf(&quot;format&quot;,arguments) [tonumber(NUMSTR,BASE)] [tostring]</td>
</tr>
<tr>
<td><strong>Cryptographic functions</strong></td>
<td>md5(X) [sha256(X) sha512(X)]</td>
</tr>
<tr>
<td><strong>Date and Time functions</strong></td>
<td>now() [strftime(X,Y) [strptime(X,Y) [time()]</td>
</tr>
<tr>
<td><strong>Informational functions</strong></td>
<td>isbool(X) [isnull(X) [isstr(X) [isnum(X) [typeof(X)]</td>
</tr>
<tr>
<td><strong>Mathematical functions</strong></td>
<td>abs(X) [ceiling(X) [exact(X) [exp(X) [floor(X) [ln(X) [log(X,Y) [pi() [pow(X,Y) | round(X,]</td>
</tr>
<tr>
<td><strong>Multivalue eval functions</strong></td>
<td>commands(X) [mvfilter(X) [mvrange(X) [mvsort(X) [mvzip(X,]</td>
</tr>
<tr>
<td><strong>Statistical eval functions</strong></td>
<td>max(X,...) [min(X,...) [random()]</td>
</tr>
<tr>
<td><strong>Text functions</strong></td>
<td>len(X) [rtrim(X,Y) [trim(X,Y) [upper(X) [urldecode]</td>
</tr>
<tr>
<td><strong>Trigonometry</strong></td>
<td>acos(X) [atan2(X,Y) [sin(X)]</td>
</tr>
<tr>
<td>sourcetype</td>
<td>Count</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------</td>
</tr>
<tr>
<td>access_combined_wcookie</td>
<td>39532</td>
</tr>
<tr>
<td>secure</td>
<td>40,000</td>
</tr>
<tr>
<td>vendor_sales</td>
<td>30,244</td>
</tr>
</tbody>
</table>
Use the **fieldformat** command to reformat the appearance of the output of these fields. The **Count** field is formatted to display the values with commas. The **First Event**, **Last Event**, and **Last Update** fields are formatted to display the values in readable timestamps.

| metadata type=sourcetypes | rename totalCount as Count firstTime as "First Event" lastTime as "Last Event" recentTime as "Last Update" | table sourcetype Count "First Event" "Last Event" "Last Update" | fieldformat Count=tostring(Count, "commas") | fieldformat "First Event"=strftime('First Event', "%c") | fieldformat "Last Event"=strftime('Last Event', "%c") | fieldformat "Last Update"=strftime('Last Update', "%c")

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>sourcetype</th>
<th>Count</th>
<th>First Event</th>
<th>Last Event</th>
<th>Last Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>secure</td>
<td>40,088</td>
<td>Mon Mar 12 00:15:01 2018</td>
<td>Tue Apr 17 00:15:06 2018</td>
<td>Wed Apr 18 09:11:16 2018</td>
</tr>
<tr>
<td>vendor_sales</td>
<td>30,244</td>
<td>Mon Mar 12 18:23:07 2018</td>
<td>Tue Apr 17 18:24:02 2018</td>
<td>Wed Apr 18 09:11:15 2018</td>
</tr>
</tbody>
</table>

**See also**

eval, where

**Date and time format variables**
Fields

Description

Keeps or removes fields from search results based on the field list criteria.

By default, the internal fields _raw and _time are included in output in Splunk Web. Additional internal fields are included in the output with the outputcsv command. See Usage.

Syntax

fields [+|-] <wc-field-list>

Required arguments

<wc-field-list>

Syntax: <string>, <string>, ...
Description: Comma-delimited list of fields to keep or remove. You can use wild card characters in the field names.

Optional arguments

+ | -

Syntax: + | -
Description: If the plus ( + ) symbol is specified, only the fields in the wc-field-list are kept in the results. If the negative ( - ) symbol is specified, the fields in the wc-field-list are removed from the results.
Default: +

Usage

The fields command is a distributable streaming command. See Command types.
**Internal fields and Splunk Web**

The leading underscore is reserved for names of internal fields such as `_raw` and `_time`. By default, the internal fields `_raw` and `_time` are included in the search results in Splunk Web. The `fields` command does not remove these internal fields unless you explicitly specify that the fields should not appear in the output in Splunk Web.

For example, to remove all internal fields, you specify:

```
... | fields - _*
```

To exclude a specific field, such as `_raw`, you specify:

```
... | fields - _raw
```

Be cautious removing the `_time` field. Statistical commands, such as `timechart` and `chart`, cannot display date or time information without the `_time` field.

**Displaying internal fields in Splunk Web**

Other than the `_raw` and `_time` fields, internal fields do not display in Splunk Web, even if you explicitly specify the fields in the search. For example, the following search does not show the `_bkt` field in the results.

```
index=_internal | head 5 | fields + _bkt | table _bkt
```

To display an internal field in the results, the field must be copied or renamed to a field name that does not include the leading underscore character. For example:

```
index=_internal | head 5 | fields + _bkt | eval bkt=_bkt | table bkt
```

**Internal fields and the outputcsv command**

When the `outputcsv` command is used in the search, there are additional internal fields that are automatically added to the CSV file. The most common internal fields that are added are:

- `_raw`
- `_time`
- `_indextime`
To exclude internal fields from the output, specify each field that you want to exclude. For example:

```
... | fields - _raw _indextime _sourcetype _serial | outputcsv MyTestCsvFile
```

**Examples**

**Example 1:**

Remove the host and ip fields from the results

```
... | fields - host, ip
```

**Example 2:**

Keep only the host and ip fields. Remove all of the internal fields. The internal fields begin with an underscore character, for example _time.

```
... | fields host, ip | fields - _*
```

**Example 3:**

Remove unwanted internal fields from the output CSV file. The fields to exclude are _raw_indextime, _sourcetype, _subsecond, and _serial.

```
index=_internal sourcetype="splunkd" | head 5 | fields - _raw _indextime _sourcetype _subsecond _serial | outputcsv MyTestCsvFile
```

**Example 4:**

Keep only the fields source, sourcetype, host, and all fields beginning with error.

```
... | fields source, sourcetype, host, error*
```

**See also**

rename, table

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the fields command.
fieldsummary

Description

The fieldsummary command calculates summary statistics for all fields or a subset of the fields in your events. The summary information is displayed as a results table.

Syntax

fieldsummary [maxvals=<num>] [<wc-field-list>]

Optional arguments

maxvals

Syntax: maxvals=<num>
Description: Specifies the maximum distinct values to return for each field.
Default: 100

wc-field-list

description: A field or list of fields. You can specify multiple, similar field names using the asterisk ( *) wildcard.

Usage

The fieldsummary command displays the summary information in a results table. The following information appears in the results table:

<table>
<thead>
<tr>
<th>Summary field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>The field name in the event.</td>
</tr>
<tr>
<td>count</td>
<td>The number of events/results with that field.</td>
</tr>
<tr>
<td>distinct_count</td>
<td>The number of unique values in the field.</td>
</tr>
<tr>
<td>is_exact</td>
<td>Whether or not the field is exact. This is related to the distinct count of the field values. If the number of values of the field exceeds maxvals, then fieldsummary will stop retaining all the values and compute an approximate distinct count instead of an exact one. 1 means it is exact, 0 means it is not.</td>
</tr>
<tr>
<td>Summary field name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>max</td>
<td>If the field is numeric, the maximum of its value.</td>
</tr>
<tr>
<td>mean</td>
<td>If the field is numeric, the mean of its values.</td>
</tr>
<tr>
<td>min</td>
<td>If the field is numeric, the minimum of its values.</td>
</tr>
<tr>
<td>numeric_count</td>
<td>The count of numeric values in the field. This would not include NULL values.</td>
</tr>
<tr>
<td>stdev</td>
<td>If the field is numeric, the standard deviation of its values.</td>
</tr>
<tr>
<td>values</td>
<td>The distinct values of the field and count of each value.</td>
</tr>
</tbody>
</table>

### Examples

1. **Return summaries for all fields**

   This example returns summaries for all fields in the `_internal` index from the last 15 minutes.

   ```
   index=_internal earliest=-15m latest=now | fieldsummary
   ```

   In this example, the results in the `max`, `min`, and `stdev` fields are formatted to display up to 4 decimal points.

2. **Return summaries for specific fields**

   This example returns summaries for fields in the `_internal` index with names that contain "size" and "count". The search returns only the top 10 values for each field from the last 15 minutes.

   ```
   index=_internal earliest=-15m latest=now | fieldsummary maxvals=10 *size* *count*
   ```
See also
analyzefields, anomalies, anomalousvalue, stats

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the fieldsummary command.

filldown

Description

Replaces null values with the last non-null value for a field or set of fields. If no list of fields is given, the fielldown command will be applied to all fields. If there are not any previous values for a field, it is left blank (NULL).

Syntax

fielldown <wc-field-list>

Required arguments

<wc-field-list>

Syntax: <string>, <string>, ...
Description: Comma-delimited list of fields to apply the fielldown command to. You can use wild card characters to specify fields with similar names.
Examples

Example 1:
Fill down null values values for all fields.

... | filldown

Example 2:
Fill down null values for the count field only.

... | filldown count

Example 3:
Fill down null values for the count field and any field that starts with 'score'.

... | filldown count score*

See also

fillna

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the filldown command.

fillna

Description

Replaces null values with a specified value. Null values are field values that are missing in a particular result but present in another result. Use fillnull to replace null field values with a string. If you do not specify a field list, fillnull replaces all null values with 0 (the default) or a user-supplied string.
Syntax

fillna [value=string] [<field-list>]

**Optional arguments**

field-list

**Syntax:** <field>...

**Description:** One or more fields, delimited with a space. If not specified, fillnull is applied to all fields.

value

**Datatype:** value=<string>

**Description:** Specify a string value to replace null values.

**Default:** 0

**Usage**

The `fillna` command is a distributable streaming command when a field-list is specified. When no field-list is specified, the fillnull command fits into the dataset processing type. See Command types.

**Examples**

**Example 1:**

For the current search results, fill all empty fields with NULL.

```plaintext
... | fillnull value=NULL
```

**Example 2:**

For the current search results, fill all empty field values of "foo" and "bar" with NULL.

```plaintext
... | fillnull value=NULL foo bar
```

**Example 3:**

For the current search results, fill all empty fields with zero.

```plaintext
... | fillnull
```
**Example 4:**

Build a time series chart of web events by host and fill all empty fields with NULL.

```
sourcetype="web" | timechart count by host | fillnull value=NULL
```

**See also**

filldown  
streamstats

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the fillnull command.

**findtypes**

**Description**

Generates suggested event types by taking the results of a search and producing a list of potential event types. At most, 5000 events are analyzed for discovering event types.

**Syntax**

```
findtypes max=<int> [notcovered] [useraw]
```

**Required arguments**

`max`

<table>
<thead>
<tr>
<th>Datatype:</th>
<th>&lt;int&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The maximum number of events to return.</td>
</tr>
<tr>
<td>Default:</td>
<td>10</td>
</tr>
</tbody>
</table>

**Optional arguments**

`notcovered`

| Description: | If this keyword is used, the findtypes command returns only event types that are not already covered. |
useraw

**Description:** If this keyword is used, the `findtypes` command uses phrases in the _raw text of events to generate event types.

**Examples**

**Example 1:**
Discover 10 common event types.

```
... | findtypes
```

**Example 2:**
Discover 50 common event types and add support for looking at text phrases.

```
... | findtypes max=50 useraw
```

**See also**

typer

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the findtypes command.

**folderize**

**Description**

Creates a higher-level grouping, such as replacing filenames with directories. Replaces the `attr` attribute value with a more generic value, which is the result of grouping the `attr` value with other values from other results, where grouping occurs by tokenizing the `attr` value on the `sep` separator value.

For example, the `folderize` command can group search results, such as those used on the Splunk Web home page, to list hierarchical buckets (e.g. directories or categories). Rather than listing 200 sources, the `folderize` command breaks the source strings by a separator (e.g. `/`) and determines if looking only at directories results in the number of results requested.
Syntax

folderize attr=<string> [sep=<string>] [size=<string>] [minfolders=<int>] [maxfolders=<int>]

Arguments

attr

Syntax: attr=<string>
Description: Replaces the attr attribute value with a more generic value, which is the result of grouping it with other values from other results, where grouping occurs by tokenizing the attribute (attr) value on the separator (sep) value.

sep

Syntax: sep=<string>
Description: Specify a separator character used to construct output field names when multiple data series are used in conjunction with a split-by field.
Default: ::

size

Syntax: size=<string>
Description: Supply a name to be used for the size of the folder.
Default: totalCount

minfolders

Syntax: minfolders=<int>
Description: Set the minimum number of folders to group.
Default: 2

maxfolders

Syntax: maxfolders=<int>
Description: Set the maximum number of folders to group.
Default: 20

Examples

1. Group results into folders based on URI

Consider this search.

index=_internal | stats count(uri) by uri

352
The following image shows the results of the search run using the All Time time range. Many of the results start with /en-US/account. Because some of the URIs are very long, the image does not show the second column on the far right. That column is the count(uri) column created by the stats command.

Using the folderize command, you can summarize the URI values into more manageable groupings.

```plaintext
index=_internal | stats count(uri) by uri | folderize size=count(uri) attr=uri sep=","/
```

The following image shows the URIs grouped in the result set.
In this example, the `count(uri)` column is the count of the unique URIs that were returned from the `stats` command. The `memberCount` column shows the count of the URIs in each group. For example, the `/en-US/` URI was found 22 times in the events, as shown in the `count(uri)` column. When the `folderize` command arranges the URI into groups, there is only 1 member in the `/en-US/` group. Whereas the URIs that start with `/services/` occurred 10088 times in the events, but there are only 1648 unique members in the `/services/*` group.

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `folderize` command.

**foreach**

**Description**

Runs a templated streaming subsearch for each field in a wildcarded field list.

**Syntax**

`foreach <wc-field>... [fieldstr=<string>] [matchstr=<string>] [matchseg1=<string>] [matchseg2=<string>] [matchseg3=<string>] <subsearch>`

**Required arguments**

- `wc-field`
  
  **Syntax:** `<field> ...`
  
  **Description:** A list of field names. You can use wild card characters in the field names.

- `subsearch`
  
  **Syntax:** `[ subsearch ]`
  
  **Description:** A subsearch that includes a template for replacing the values of the wildcarded fields.

**Optional arguments**

- `fieldstr`
  
    **Syntax:** `fieldstr=<string>
    
    **Description:** Replaces the `<<FIELD>>` token with the whole field name.
matchstr
   **Syntax:** matchstr=<string>
   **Description:** Replaces <<MATCHSTR>> with part of the field name that matches wildcard(s) in the specifier.

matchseg1
   **Syntax:** matchseg1=<string>
   **Description:** Replaces <<MATCHSEG1>> with part of the field name that matches the first wildcard.

matchseg2
   **Syntax:** matchseg2=<string>
   **Description:** Replaces <<MATCHSEG2>> with part of the field name that matches the second wildcard.

matchseg3
   **Syntax:** matchseg3=<string>
   **Description:** Replaces <<MATCHSEG3>> with part of the field name that matches the third wildcard.

**Usage**

If the field names contain characters other than alphanumeric characters, such as dashes, underscores, or periods, you need to enclose the <<FIELD>> token in single quotation marks in the `eval` command portion of the search.

For example, the following search adds the values from all of the fields that start with similar names.

```plaintext
... | eval total=0 | eval test_1=1 | eval test_2=2 | eval test_3=3 | foreach test* [eval total=total + '<<FIELD>>']
```

The <<FIELD>> token in the `foreach` subsearch is just a string replacement of the field names `test*`. The `eval` expression does not recognize field names with non-alphanumeric characters unless the field names are surrounded by single quotation marks. For the `eval` expression to work, the <<FIELD>> token needs to be surrounded by single quotation marks.

**Examples**
1. Add the values from all of the fields that start with similar names

The following search adds the values from all of the fields that start with similar names. You can run this search on your own Splunk instance.

```splunk
|makeresults 1| eval total=0 | eval test1=1 | eval test2=2 | eval test3=3 | foreach test* [eval total=total + <<FIELD>>]
```

- This search creates 1 result using the `makeresults` command.
- The search then uses the `eval` command to create the fields `total`, `test1`, `test2`, and `test3` with corresponding values.
- The `foreach` command is used to perform the subsearch for every field that starts with "test". Each time the subsearch is run, the previous total is added to the value of the test field to calculate the new total. The final total after all of the "test" fields are processed is 6.

The following table shows how the subsearch iterates over each "test" field. The table shows the beginning value of the "total" field each time the subsearch is run and the calculated total based on the value for the "test" field.

<table>
<thead>
<tr>
<th>Subsearch iteration</th>
<th>Test field</th>
<th>Total field start value</th>
<th>Test field value</th>
<th>Calculation of &quot;total&quot; field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>test1</td>
<td>0</td>
<td>1</td>
<td>0+1=1</td>
</tr>
<tr>
<td>2</td>
<td>test2</td>
<td>1</td>
<td>2</td>
<td>1+2=3</td>
</tr>
<tr>
<td>3</td>
<td>test3</td>
<td>3</td>
<td>3</td>
<td>3+3=6</td>
</tr>
</tbody>
</table>

2. Monitor license usage

Use the `foreach` command to monitor license usage.

First run the following search on the license master to return the daily license usage per sourcetype in bytes:

```splunk
index=_internal source=*license_usage.log type!='*Summary' earliest=-30d | timechart span=1d sum(b) AS daily_bytes by st
```

Use the `foreach` command to calculate the daily license usage in gigabytes for each field:

```splunk
index=_internal source=*license_usage.log type!='*Summary' earliest=-30d | timechart span=1d sum(b) AS daily_bytes by st | foreach * [eval <<FIELD>>='<<FIELD>>'/1024/1024/1024]
```
3. Use the <<MATCHSTR>>

Add each field that matches foo* to the corresponding bar* and write the result to a new_* field. For example, new_X = fooX + barX.

... | foreach foo* [eval new_<<MATCHSTR>> = <<FIELD>> + bar<<MATCHSTR>>]

4.

Equivalent to ... | eval foo="foo" | eval bar="bar" | eval baz="baz"

... | foreach foo bar baz [eval <<FIELD>> = "<<FIELD>>"]

5.

For the field, fooXbarY, this is equivalent to: ... | eval fooXbarY = "Y"

... | foreach foo*bar* fieldstr="#field#" matchseg2="#matchseg2#" [eval #field# = "#matchseg2#"]

See also
eval, map

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the foreach command.

format

Description

This command is used implicitly by subsearches. This command takes the results of a subsearch, formats the results into a single result and places that result into a new field called search.
Syntax

format [mvsep="<mv separator>"] [maxresults=<int>] ["<row prefix>" "<column prefix>" "<column separator>" "<column end>" "<row separator>" "<row end>"]

If you want to specify a row or column options, you must specify all of the row and column options.

Optional arguments

mvsep
   Syntax: mvsep="<string>"
   Description: The separator to use for multivalue fields.
   Default: OR

maxresults
   Syntax: maxresults=<int>
   Description: The maximum results to return.
   Default: 0, which means no limitation on the number of results returned.

<row prefix>
   Syntax: "<string>"
   Description: The value to use for the row prefix.
   Default: The open parenthesis character "(

<column prefix>
   Syntax: "<string>"
   Description: The value to use for the column prefix.
   Default: The open parenthesis character "(

<column separator>
   Syntax: "<string>"
   Description: The value to use for the column separator.
   Default: AND

<column end>
   Syntax: "<string>"
   Description: The value to use for the column end.
   Default: The close parenthesis character ")"
Description: The value to use for the row separator.
Default: OR

Syntax: "<string>"
Description: The value to use for the column end.
Default: The close parenthesis character ")

Usage
By default, when you do not specify any of the optional row and column arguments, the output of the format command defaults to: "(" "(" "AND" ")" 
"OR" ")".

The only reason to specify the row and column arguments is to export the query to another system that requires different formatting.

Examples

1. Example with no optional parameters

Suppose that you have results that look like this:

<table>
<thead>
<tr>
<th>source</th>
<th>sourcetype</th>
<th>host</th>
</tr>
</thead>
<tbody>
<tr>
<td>syslog.log</td>
<td>syslog</td>
<td>my_laptop</td>
</tr>
<tr>
<td>bob-syslog.log</td>
<td>syslog</td>
<td>bobs_laptop</td>
</tr>
<tr>
<td>laura-syslog.log</td>
<td>syslog</td>
<td>lauras_laptop</td>
</tr>
</tbody>
</table>

The following search returns the top 2 results, and creates a search based on the host, source, and sourcetype fields. The default format settings are used.

... | head 2 | fields source, sourcetype, host | format

This search returns the syntax for a search that is based on the field values in the top 2 results. The syntax is placed into a new field called search.

<table>
<thead>
<tr>
<th>source</th>
<th>sourcetype</th>
<th>host</th>
<th>search</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>( ( host=&quot;my_laptop&quot; AND source=&quot;syslog.log&quot; AND sourcetype=&quot;syslog&quot; ) OR ( host=&quot;bobs_laptop&quot; AND source=&quot;bob-syslog.log&quot; AND sourcetype=&quot;syslog&quot; ) )</td>
</tr>
</tbody>
</table>
### 2. Example using the optional parameters

You want to produce output that is formatted to use on an external system.

```
... | format "[ "[ "&" ""] "||" "]"
```

Using the data in Example 1, the result is:

```
source sourcetype host search
| [ [ host="mylaptop" && source="syslog.log" && sourcetype="syslog" ] | [ host="bobslaptop" && source="bob-syslog.log" && sourcetype="syslog" ] ]
```

### 3. Multivalue separator example

The following search uses the `eval` command to create a field called "foo" that contains one value "eventtype,log_level". The `makemv` command is used to make the foo field a multivalue field and specifies the comma as the delimiter between the values. The search then outputs only the foo field and formats that field.

```
index=_internal |head 1 |eval foo="eventtype,log_level" | makemv delim="", foo | fields foo | format mvsep="mvseparator" "{ "[ "AND" " ]" "AND" " }"
```

This results in the following output:

```
foo search
| [ ( foo="eventtype" mvseparator foo="log_level" ) ]
```

### See also

- search

### Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the format command.
from

Description

The `from` command retrieves data from a dataset, such as a data model dataset, a CSV lookup, a KV Store lookup, a saved search, or a table dataset.

Design a search that uses the `from` command to reference a dataset. Optionally add additional SPL such as lookups, eval expressions, and transforming commands to the search. Save the result as a report, alert, or dashboard panel. If you use Splunk Cloud, or use Splunk Enterprise and have installed the Splunk Datasets Add-on, you can also save the search as a table dataset.

See the Usage section.

Syntax

```
| from <dataset_type>:<dataset_name>
```

Required arguments

<dataset_type>

**Syntax:** `<dataset_type>`

**Description:** The type of dataset. Valid values are: `datamodel`, `lookup`, and `savedsearch`. The `datamodel` dataset type can be either a data model dataset or a table dataset. You create data model datasets with the Data Model Editor. You can create table datasets with the Table Editor if you use Splunk Cloud or use Splunk Enterprise and have installed the Splunk Datasets Add-on. The `lookup` dataset type can be either a CSV lookup or a KV Store lookup. The `savedsearch` dataset type is a saved search. You can use `from` to reference any saved search as a dataset. See About datasets in the Knowledge Manager Manual.

<dataset_name>

**Syntax:** `<dataset_name>`

**Description:** The name of the dataset that you want to retrieve data from. If the `dataset_type` is a data model, the syntax is `<datamodel_name>.<dataset_name>`. If the name of the dataset contains spaces, enclose the dataset name in quotation marks.

**Example:** If the data model name is `internal_server`, and the dataset name is `splunkdaccess`, specify `internal_server.splunkdaccess` for the
In older versions of the Splunk software, the term "data model object" was used. That term has been replaced with "data model dataset".

**Optional arguments**

None.

**Usage**

The `from` command is a generating command. It can be either report-generating or event-generating depending on the search or knowledge object that is referenced by the command. See Command types.

Generating commands use a leading pipe character and should be the first command in a search. However, you can use the `from` command inside the `append` command.

When you use the `from` command, you must reference an existing dataset. You can reference any dataset listed in the Datasets listing page, such as data model datasets, CSV lookup files, CSV lookup definitions, and table datasets. You can also reference saved searches and KV Store lookup definitions. See View and manage datasets in the Knowledge Manager Manual.

**Knowledge object dependencies**

When you create a knowledge object such as a report, alert, dashboard panel, or table dataset, that knowledge object has a dependency on the referenced dataset. This is referred to as a dataset extension. When you make a change to the original dataset, such as removing or adding fields, that change propagates down to the reports, alerts, dashboard panels, and tables that have been extended from that original dataset. See Dataset extension in the Knowledge Manager Manual.

**When field filtering is disabled for a data model**

When you search the contents of a data model using the `from` command, by default the search returns a strictly-filtered set of fields. It returns only default fields and fields that are explicitly identified in the constraint search that defines the data model.
If you have edit access to your local `datamodel.conf` file, you can disable field filtering for specific data models by adding the `strict_fields=false` setting to their stanzas. When you do this, searches of data models with that setting return all fields related to the data model, including fields inherited from parent data models, fields extracted at search time, calculated fields, and fields derived from lookups.

**Examples**

**1. Search a data model**

Search a data model that contains internal server log events for REST API calls. In this example, `internal_server` is the data model name and `splunkdaccess` is the dataset inside the `internal_server` data model.

```
| from datamodel:internal_server.splunkdaccess
```

**2. Search a lookup file**

Search a lookup file that contains geographic attributes for each country, such as continent, two-letter ISO code, and subregion.

```
| from lookup:geo_attr_countries.csv
```

**3. Retrieve data by using a lookup file**

Search the contents of the KV store collection `kvstorecoll` that have a `CustID` value greater than 500 and a `CustName` value that begins with the letter P. The collection is referenced in a lookup table called `kvstorecoll_lookup`. Using the `stats` command, provide a count of the events received from the table.

```
| from lookup:kvstorecoll_lookup | where (CustID>500) AND (CustName="P*") | stats count
```

**4. Retrieve data using a saved search**

Retrieve the timestamp and client IP from the saved search called `mysecurityquery`.

```
| from savedsearch:mysecurityquery | fields _time clientip ...
```
5. Specify a dataset name that contains spaces

When the name of a dataset includes spaces, enclose the dataset name in quotation marks.

| from savedsearch: "Top five sourcetypes"

See also

lookup, datamodel

gauge

Description

Use the `gauge` command to transform your search results into a format that can be used with the gauge charts. Gauge charts are a visualization of a single aggregated metric, such as a count or a sum.

The output of the `gauge` command is a single numerical value stored in a field called `x`. You can specify a range to display in the gauge or use the default range of 0 to 100.

For more information about using the `gauge` command with the gauge chart types, see Using gauges in the Gauges section in *Dashboards and Visualizations*.

Syntax

```
gauge <value> [range_val1> <range_val2> ...]
```

Required arguments

value

Syntax: field_name | <num>

Description: A numeric field or literal number to use as the current value of the gauge. If you specify a numeric field, the `gauge` command uses the first value in that field as the value for the gauge.
Optional arguments

range values

Syntax: <range_val1> <range_val2> ...
Description: A space-separated list of two or more numeric fields or numbers to use as the overall numeric range displayed in the gauge. Each range value can be a numeric field name or a literal number. If you specify a field name, the first value in that field is used as the range value. The total range of the gauge is from the first range_val to the last range_val.
See Usage.
Default range: 0 to 100

Usage

You can create gauge charts without using the gauge command as long as your search results in a single value. The advantage of using the gauge command is that you can specify a set of range values instead of using the default range values of 0 to 100.

Specifying ranges

If you specify range values, you must specify at least two values. The gauge begins at the first value and ends at the last value that you specify.

If you specify more than two range_val arguments, the intermediate range values are used to split the total range into subranges. Each subrange displays in different color, which creates a visual distinction.

The range values are returned as a series of fields called y1, y2, and so on.

If you do not specify range values, the range defaults to a low value of 0 and a high value of 100.

If a single range value is specified, it is ignored.

Gauge colors

With a gauge chart, a single numerical value is mapped against a set of colors. These colors can have particular business meaning or business logic. As the value changes over time, the gauge marker changes position within this range.

The color ranges in the gauge chart are based on the range values that you specify with the gauge command. When you specify range values, you define the
overall numerical range represented by the gauge. You can define the size of the colored bands within that range. If you want to use the color bands, add four range values to the search string. These range values indicate the beginning and end of the range. These range values also indicate the relative sizes of the color bands within this range.

**Examples**

1. **Create a gauge with multiple ranges**

Count the number of events and display the count on a gauge with four ranges, from 0-750, 750-1000, 1000-1250, and 1250-1500.

Start by generating the results table using this search. Run the search using the **Last 15 minutes** time range.

    index=_internal | stats count as myCount | gauge myCount 750 1000 1250 1500

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>x</th>
<th>y1</th>
<th>y2</th>
<th>y3</th>
<th>y4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3321</td>
<td>750</td>
<td>1000</td>
<td>1250</td>
<td>1500</td>
</tr>
</tbody>
</table>

Click on the **Visualizations** tab. There are three types of gauges that you can choose from: radial, filler, and marker. The following image shows the radial gauge that is created based on the search results.
For more information about using the `gauge` command with the gauge chart type, see the Gauges section in *Dashboard and Visualizations*.

**See also**

- **Commands**
  - `eval`, `stats`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `gauge` command.

---

**gentimes**

**Description**

The `gentimes` command is useful in conjunction with the `map` command.

Generates timestamp results starting with the exact time specified as start time. Each result describes an adjacent, non-overlapping time range as indicated by the increment value. This terminates when enough results are generated to pass the endtime value.

This command does not work for future dates.

**Syntax**

```
| gentimes start=<timestamp> [end=<timestamp>] [increment=<increment>]
```

- **Required arguments**
  - `start`
    - **Syntax:** `start=<timestamp>`
    - **Description:** Specify as start time.
  - `<timestamp>`
    - **Syntax:** `MM/DD/YYYY[:HH:MM:SS] | <int>`
    - **Description:** Indicate the timeframe, for example: `10/1/2017` for October 1, 2017, `4/1/2017:12:34:56` for April 1, 2017 at 12:34:56, or `-5` for five days ago.
Optional arguments

end
Syntax: end=<timestamp>
Description: Specify an end time.
Default: midnight, prior to the current time in local time

increment
Syntax: increment=<int>(s | m | h | d)
Description: Specify a time period to increment from the start time to the end time. Supported increments are seconds, minutes, hours, and days.
Default: 1d

Usage

The gentimes command is an event-generating command. See Command types.

Generating commands use a leading pipe character and should be the first command in a search.

The gentimes command returns four fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>starttime</td>
<td>The starting time range in UNIX time.</td>
</tr>
<tr>
<td>starthuman</td>
<td>The human readable time range in the format DDD MMM DD HH:MM:SS YYYY. For example Sun Apr 1 00:00:00 2018.</td>
</tr>
<tr>
<td>endtime</td>
<td>The ending time range in UNIX time.</td>
</tr>
</tbody>
</table>

Examples

1. Generate daily time ranges by specifying dates

Generates daily time ranges from April 1 to April 5 in 2018. This search generates four intervals covering one day periods aligning with the calendar days April 1, 2, 3, and 4, during 2018.

| gentimes start=4/1/18 end=4/5/18 |

The results appear on the Statistics tab and look something like this:
<table>
<thead>
<tr>
<th>starttime</th>
<th>starthuman</th>
<th>endtime</th>
<th>endhuman</th>
</tr>
</thead>
<tbody>
<tr>
<td>1522566000</td>
<td>Sun Apr 1 00:00:00 2018</td>
<td>1522652399</td>
<td>Sun Apr 1 23:59:59 2018</td>
</tr>
<tr>
<td>1522652400</td>
<td>Mon Apr 2 00:00:00 2018</td>
<td>1522738799</td>
<td>Mon Apr 2 23:59:59 2018</td>
</tr>
<tr>
<td>1522738800</td>
<td>Tue Apr 3 00:00:00 2018</td>
<td>1522825199</td>
<td>Tue Apr 3 23:59:59 2018</td>
</tr>
<tr>
<td>1522825200</td>
<td>Wed Apr 4 00:00:00 2018</td>
<td>1522911599</td>
<td>Wed Apr 4 23:59:59 2018</td>
</tr>
</tbody>
</table>

### 2. Generate daily time ranges by specifying relative times

Generate daily time ranges from 30 days ago until 27 days ago.

```
gentimes start=-30 end=-27
```

### 3. Generate hourly time ranges

Generate hourly time ranges from December 1 to December 5 in 2017.

```
gentimes start=12/1/17 end=12/5/17 increment=1h
```

### 4. Generate time ranges by only specifying a start date

Generate daily time ranges from September 25 to today.

```
gentimes start=9/25/17
```

### 5. Generate weekly time ranges

Although the week increment is not supported, you can generate a weekly increment by specifying `increment=7d`.

This example generates weekly time ranges from December 1, 2017 to April 30, 2018.

```
gentimes start=12/1/17 end=4/30/18 increment=7d
```

### See also

makeresults, map
**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the gentimes command.

**geom**

**Description**

The `geom` command adds a field, named `geom`, to each result. This field contains geographic data structures for polygon geometry in JSON. These geographic data structures are used to create choropleth map visualizations.

![Map](image)

For more information about choropleth maps, see Mapping data in the *Dashboards and Visualizations* manual.

**Syntax**

```
geom [<featureCollection>] [allFeatures=<boolean>] [featureIdField=<string>]
[gen=<double>] [min_x=<double>] [min_y=<double>] [max_x=<double>] [max_y=<double>]
```

**Required arguments**

None.

**Optional arguments**

- **featureCollection**
  - **Syntax:** `<geo_lookup>`
  - **Description:** Specifies the geographic lookup file that you want to use. Two geographic lookup files are included by default with Splunk software: `geo_us_states` and `geo_countries`. You can install your own geographic lookups from KMZ or KLM files. See **Usage** for more information.
allFeatures
  Syntax: allFeatures=<bool>
  Description: Specifies that the output include every geometric feature in
  the feature collection. When a shape has no values, any aggregate fields,
  such as average or count, display zero when this argument is used.
  Additional rows are appended for each feature that is not already present
  in the search results when this argument is used. See Examples.
  Default: false

featureIdField
  Syntax: featureIdField=<field>
  Description: If the event contains the featureId in a field named
  something other than "featureId", use this option to specify the field name.

gen
  Syntax: gen=<double>
  Description: Specifies generalization, in the units of the data. For
  example, gen=0.1 generalizes, or reduces the size of, the geometry by
  running the Douglass Puiker Ramer algorithm on the polygons with a
  parameter of 0.1 degrees.
  Default: 0.1

min_x
  Syntax: min_x=<double>
  Description: The X coordinate for the bottom-left corner of the bounding
  box for the geometric shape. The range for the coordinate is -180 to 180.
  See Usage for more information.
  Default: -180

min_y
  Syntax: min_y=<double>
  Description: The Y coordinate for the bottom-left corner of the bounding
  box for the geometric shape. The range for the coordinate is -90 to 90.
  Default: -90

max_x
  Syntax: max_x=<double>
  Description: The X coordinate for the upper-right corner of the bounding
  box for the geometric shape. The range for the coordinate -180 to 180.
  Default: 180

max_y
  Syntax: max_y=<double>
Description: The Y coordinate for the upper-right corner of the bounding box for the geometric shape. The range is -90 to 90.
Default: 90

Usage

Specifying a lookup

To use your own lookup file, you can define the lookup in Splunk Web or edit the transforms.conf file.

If you use a managed Splunk Cloud deployment you must use Splunk Web to define a lookup.

Define a geospatial lookup in Splunk Web

1. To create a geospatial lookup in Splunk Web, you use the Lookups option in the Settings menu. You must add the lookup file, create a lookup definition, and can set the lookup to work automatically. See Define a geospatial lookup in Splunk Web in the Knowledge Manager Manual.

Configure a geospatial lookup in transforms.conf

1. Edit the %SPLUNK_HOME%/etc/system/local/transforms.conf file, or create a new file named transforms.conf in the %SPLUNK_HOME%/etc/system/local directory, if the file does not already exist. See How to edit a configuration file in the Admin Manual.
2. Specify the name of the lookup stanza in the transforms.conf file for the featureCollection argument.

Specifying no optional arguments

When no arguments are specified, the geom command looks for a field named featureCollection and a field named featureIdField in the event. These fields are present in the default output from a geoindex lookup.

Clipping the geometry

The min_x, min_y, max_x, and max_y arguments are used to clip the geometry. Use these arguments to define a bounding box for the geometric shape. You can specify the minimum rectangle corner (min_x, min_y) and the maximum rectangle
corner \((\text{max}_x, \text{max}_y)\). By specifying the coordinates, you are returning only the data within those coordinates.

**Testing lookup files**

You can use the `inputlookup` command to verify that the geometric features on the map are correct. The syntax is `| inputlookup <your_lookup>`.

For example, to verify that the geometric features in built-in `geo_us_states` lookup appear correctly on the choropleth map:

1. Run the following search:

   ```bash
   | inputlookup geo_us_states
   ```

2. On the **Visualizations** tab, change to a Choropleth Map.
3. Zoom in to see the geometric features. In this example, the states in the United States.

**Testing geometric features**

You can create an arbitrary result to test the geometric features.

To show how the output appears with the `allFeatures` argument, the following search creates a simple set of fields and values.

```bash
| stats count | eval featureId="California" | eval count=10000 | geom geo_us_states allFeatures=true
```

- The search uses the `stats` command, specifying the `count` field. A single result is created that has a value of zero (0) in the `count` field.
- The `eval` command is used to add the `featureId` field with value of California to the result.
- Another `eval` command is used to specify the value 10000 for the `count` field. You now have a single result with two fields, `count` and `featureId`.
- When the `geom` command is added, two additional fields are added, `featureCollection` and `geom`.

The following image shows the results of the search on the **Statistics** tab.

373
The following image shows the results of the search on the Visualization tab. Make sure that the map is a Choropleth Map. This image is zoomed in to show more detail.
Examples

1. Use the default settings

When no arguments are provided, the `geom` command looks for a field named `featureCollection` and a field named `featureId` in the event. These fields are present in the default output from a geospatial lookup.

```bash
...| geom
```

2. Use the built-in geospatial lookup

This example uses the built-in `geo_us_states` lookup file for the `featureCollection`.

```bash
...| geom geo_us_states
```

3. Specify a field that contains the `featureId`

This example uses the built-in `geo_us_states` lookup and specifies `state` as the `featureIdField`. In most geospatial lookup files, the feature IDs are stored in a field called `featureId`. Use the `featureIdField` argument when the event contains the feature IDs in a field named something other than "featureId".

```bash
...| geom geo_us_states featureIdField="state"
```

4. Show all geometric features in the output

The following example specifies that the output include every geometric feature in the feature collection. If no value is present for a geometric feature, zero is the default value. Using the `allFeatures` argument causes the choropleth map visualization to render all of the shapes.

```bash
...| geom geo_us_states allFeatures=true
```

5. Use the built-in countries lookup

The following example uses the built-in `geo_countries` lookup. This search uses the `lookup` command to specify shorter field names for the latitude and longitude fields. The `stats` command is used to count the feature IDs and renames the `featureIdField` field as `country`. The `geom` command generates the information for the choropleth map using the renamed field `country`. 
6. Specify the bounding box for the geometric shape

This example uses the `geom` command attributes that enable you to clip the geometry by specifying a bounding box.

```bash
... | geom geo_us_states featureIdField="state" gen=0.1 min_x=-130.5 min_y=37.6 max_x=-130.1 max_y=37.7
```

See also

Mapping data in the *Dashboards and Visualizations* manual.

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the geom command.

**geomfilter**

**Description**

Use the geomfilter command to specify points of a bounding box for clipping choropleth maps.

For more information about choropleth maps, see "Mapping data" in the *Dashboards and Visualizations* Manual.

**Syntax**

```
geomfilter [min_x=<float>] [min_y=<float>] [max_x=<float>] [max_y=<float>]
```

**Optional arguments**

- **min_x**
  - **Syntax:** `min_x=<float>`
  - **Description:** The x coordinate of the bounding box’s bottom-left corner, in the range [-180, 180].
  - **Default:** -180
min_y

Syntax:  min_y=<float>
Description: The y coordinate of the bounding box's bottom-left corner, in the range [-90, 90].
Default: -90

max_x

Syntax:  max_x=<float>
Description: The x coordinate of the bounding box's up-right corner, in the range [-180, 180].
Default: 180

max_y

Syntax:  max_y=<float>
Description: The y coordinate of the bounding box's up-right corner, in the range [-90, 90].
Default: max_y=90

Usage

The geomfilter command accepts two points that specify a bounding box for clipping choropleth maps. Points that fall outside of the bounding box will be filtered out.

Examples

Example 1: This example uses the default bounding box, which will clip the entire map.

```bash
...| geomfilter
```

Example 2: This example clips half of the whole map.

```bash
...| geomfilter min_x=-90 min_y=-90 max_x=90 max_y=90
```
See also

gem

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the geomfilter command.

geostats

Description

Use the `geostats` command to generate statistics to display geographic data and summarize the data on maps.

The command generates statistics which are clustered into geographical bins to be rendered on a world map. The events are clustered based on latitude and longitude fields in the events. Statistics are then evaluated on the generated clusters. The statistics can be grouped or split by fields using a `BY` clause.

For map rendering and zooming efficiency, the `geostats` command generates clustered statistics at a variety of zoom levels in one search, the visualization selecting among them. The quantity of zoom levels is controlled by the `binspanlat`, `binspanlong`, and `maxzoomlevel` options. The initial granularity is selected by the `binspanlat` and the `binspanlong`. At each level of zoom, the number of bins is doubled in both dimensions for a total of 4 times as many bins for each zoom in.

Syntax

```
geostats [translatetoxy=<bool>] [latfield=<string>] [longfield=<string>]
[globalimit=<int>] [locallimit=<int>] [outputlatfield=<string>]
[outputlongfield=<string>] [ binspanlat=<float> binspanlong=<float> ]
[maxzoomlevel=<int>] <stats-agg-term>... [by-clause]
```

**Required arguments**

`stats-agg-term`

**Syntax:** `<stats-func> ( <evaled-field> | <wc-field> ) [AS <wc-field>]`
**Description**: A statistical aggregation function. See [Stats function options](#). The function can be applied to an eval expression, or to a field or set of fields. Use the AS clause to place the result into a new field with a name that you specify. You can use wild card characters in field names. For more information on eval expressions, see Types of eval expressions in the Search Manual.

**Optional arguments**

**binspanlat**
- **Syntax**: `binspanlat=<float>`
- **Description**: The size of the bins in latitude degrees at the lowest zoom level.
- **Default**: 22.5. If the default values for `binspanlat` and `binspanlong` are used, a grid size of 8x8 is generated.

**binspanlong**
- **Syntax**: `binspanlong=<float>`
- **Description**: The size of the bins in longitude degrees at the lowest zoom level.
- **Default**: 45.0. If the default values for `binspanlat` and `binspanlong` are used, a grid size of 8x8 is generated.

**by-clause**
- **Syntax**: `BY <field>`
- **Description**: The name of the field to group by.

**globallimit**
- **Syntax**: `globallimit=<int>`
- **Description**: Controls the number of named categories to add to each pie chart. There is one additional category called "OTHER" under which all other split-by values are grouped. Setting `globallimit=0` removes all limits and all categories are rendered. Currently the grouping into "OTHER" only works intuitively for count and additive statistics.
- **Default**: 10

**locallimit**
- **Syntax**: `locallimit=<int>`
- **Description**: Specifies the limit for series filtering. When you set `locallimit=N`, the top N values are filtered based on the sum of each series. If `locallimit=0`, no filtering occurs.
- **Default**: 10
latfield

**Syntax:** latfield=<field>

**Description:** Specify a field from the pre-search that represents the latitude coordinates to use in your analysis.

**Defaults:** lat

longfield

**Syntax:** longfield=<field>

**Description:** Specify a field from the pre-search that represents the longitude coordinates to use in your analysis.

**Default:** lon

maxzoomlevel

**Syntax:** maxzoomlevel=<int>

**Description:** The maximum number of levels to create in the quadtree.

**Default:** 9. Specifies that 10 zoom levels are created, 0-9.

outputlatfield

**Syntax:** outputlatfield=<string>

**Description:** Specify a name for the latitude field in your geostats output data.

**Default:** latitude

outputlongfield

**Syntax:** outputlongfield=<string>

**Description:** Specify a name for the longitude field in your geostats output data.

**Default:** longitude

translatetoxy

**Syntax:** translatetoxy=<bool>

**Description:** If true, geostats produces one result per each locationally binned location. This mode is appropriate for rendering on a map. If false, geostats produces one result per category (or tuple of a multiply split dataset) per locationally binned location. Essentially this causes the data to be broken down by category. This mode cannot be rendered on a map.

**Default:** true

**Stats function options**

stats-func

**Syntax:** The syntax depends on the function that you use. Refer to the table below.
Description: Statistical and charting functions that you can use with the `geostats` command. Each time you invoke the `geostats` command, you can use one or more functions.

The following table lists the supported functions by type of function. Use the links in the table to see descriptions and examples for each function. For an overview about using functions with commands, see Statistical and charting functions.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate functions</td>
<td><code>avg()</code> <code>count()</code> <code>distinct_count()</code> <code>estdc()</code> <code>estdc_error()</code> <code>exactperc&lt;int&gt;()</code> <code>max()</code> <code>median()</code> <code>min()</code> <code>mode()</code> <code>perc&lt;int&gt;()</code> <code>range()</code> <code>stdev()</code> <code>stdevp()</code> <code>sum()</code> <code>sumsq()</code> <code>upperperc&lt;int&gt;()</code> <code>var()</code> <code>varp()</code></td>
</tr>
<tr>
<td>Event order functions</td>
<td><code>earliest()</code> <code>first()</code> <code>last()</code> <code>latest()</code></td>
</tr>
<tr>
<td>Multivalue stats and chart functions</td>
<td><code>list(X)</code> <code>values(X)</code></td>
</tr>
</tbody>
</table>

Usage

To display the information on a map, you must run a reporting search with the `geostats` command.

If you are using a `lookup` command before the `geostats` command, see Optimizing your lookup search.

Memory and maximum results

In the `limits.conf` file, the `maxresultrows` setting in the `[searchresults]` stanza specifies the maximum number of results to return. The default value is 50,000. Increasing this limit can result in more memory usage.
The `max_mem_usage_mb` setting in the `[default]` stanza is used to limit how much memory the `geostats` command uses to keep track of information. If the `geostats` command reaches this limit, the command stops adding the requested fields to the search results. You can increase the limit, contingent on the available system memory.

If you are using Splunk Cloud and want to change either of these limits, file a Support ticket.

**Basic examples**

1. **Use the default settings and calculate the count**

   Cluster events by default latitude and longitude fields "lat" and "lon" respectively. Calculate the count of the events.

   ```
   ... | geostats count
   ```

2. **Specify the latfield and longfield and calculate the average of a field**

   Compute the average rating for each gender after clustering/grouping the events by "eventlat" and "eventlong" values.

   ```
   ... | geostats latfield=eventlat longfield=eventlong avg(rating) by gender
   ```

**Extended examples**

3. **Count each product sold by a vendor and display the information on a map**

   This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range `All time` when you run the search.

   In addition, this example uses several lookup files that you must download (prices.csv.zip and vendors.csv.zip) and unzip the files. You must complete the steps in the Enabling field lookups section of the tutorial for both the `prices.csv` and the `vendors.csv` files. The steps in the tutorial are specific to the `prices.csv` file. For the `vendors.csv` file, use the name `vendors_lookup` for the lookup definition. Skip the step in the tutorial that makes the lookups automatic.
This search uses the `stats` command to narrow down the number of events that the `lookup` and `geostats` commands need to process.

Use the following search to count each product sold by a vendor and display the information on a map.

```
sourcetype=vendor_sales | stats count by Code VendorID | lookup prices_lookup Code OUTPUTNEW product_name | table product_name VendorID | lookup vendors_lookup VendorID | geostats latfield=VendorLatitude longfield=VendorLongitude count by product_name
```

- In this example, `sourcetype=vendor_sales` is associated with a log file that is included in the Search Tutorial sample data. This log file contains vendor information that looks like this:

  ```
  AcctID=6024298300471575
  ```

- The `vendors_lookup` is used to output all the fields in `vendors.csv` file that match to the `VendorID` in the `vendor_sales.log` file. The fields in the `vendors.csv` file are: Vendor, VendorCity, VendorID, VendorLatitude, VendorLongitude, VendorStateProvince, and VendorCountry.
- The `prices_lookup` is used to match the `Code` field in each event to a `product_name` in the table.

In this example, the CSV files are uploaded and the lookups are defined but are not automatic.

This search produces a table displayed on the **Statistics** tab:
Click the **Visualization** tab. The results are plotted on a world map. There is a pie chart for each vendor in the results. The larger the pie chart, the larger the count value.

In this screen shot, the mouse pointer is over the pie chart for a region in the northeastern part of the United States. An popup information box displays the latitude and longitude for the vendor, as well as a count of each product that the vendor sold.

You can zoom in to see more details on the map.

**See also**

iplocation, stats, xyseries

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the geostats command.

**head**

**Description**

Returns the first N number of specified results in search order. This means the most recent N events for a historical search, or the first N captured events for a real-time search. The search results are limited to the first results in search order.
There are two types of limits that can be applied: an absolute number of results, or an expression where all results are returned until the expression becomes false.

If no options or limits are explicitly stated, the `head` command returns the first 10 results.

If a numeric limit such as a numeric literal or the flag `limit=int` is used, the `head` command returns the first N results where N is the selected number. Using both numeric limit syntaxes results in an error.

If an `eval` expression is used, all initial results are returned until the first result where the expression evaluates as false. In this case, no results are returned. The result where the expression evaluates as false is kept or dropped in accordance with the `keeplast` option.

If both a numeric limit and an `eval` expression are used, the smaller of the two constraints applies. For example

```plaintext
... |head limit=10 (1==1)
```

returns up to the first 10 results, because the `eval` expression is always true. However,

```plaintext
... |head limit=10 (0==1)
```

returns no results, because the `eval` expression is always false.

**Syntax**

`head [<N> | (<eval-expression>)] [limit=<int>] [null=<bool>] [keeplast=<bool>]

*Required arguments*

None.

If no options or limits are specified, the `head` command returns the first 10 results.

*Optional arguments*

*<N>*

**Syntax:** `<int>`

**Description:** The number of results to return.
Default: 10

limit

**Syntax:** limit=<int>
**Description:** Another way to specify the number of results to return.
**Default:** 10

eval-expression

**Syntax:** <eval-compare-exp> | <eval-bool-exp>
**Description:** A valid eval expression that evaluates to a Boolean. The search returns results until this expression evaluates to false. For more information, see the evaluation functions in the *Search Reference*.

keeplast

**Syntax:** keeplast=<bool>
**Description:** Use in conjunction with the eval-expression argument to determine whether the last result in the result set is retained. The last result returned is the result that caused the eval-expression to evaluate to false or NULL. Set keeplast to true to retain the last result in the result set. Set keeplast to false to discard the last result.
**Default:** true

null

**Syntax:** null=<bool>
**Description:** Use in conjunction with the eval-expression argument to specify how a NULL result is treated. For example, if the eval expression is \((x > 10)\) and the field x does not exist, the expression evaluates to NULL instead of true or false. Set null to true to have the head command continue when it gets a NULL result. Set null to false to have the head command stop when it when it gets a NULL result.
**Default:** false

Usage

The `head` command is a centralized streaming command. See *Command types*.

Examples

1. **Return a specific number of results**

Return the first 20 results.

```bash
... | head 20
```
2. Return results based on a specified limit

Return events until the time span of the data is $\geq 100$ seconds

$$
\ldots \mid \text{streamstats range(_time) as timerange} \mid \text{head (timerange<100)}
$$

See also

reverse, tail

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the head command.

highlight

Description

Highlights specified terms in the events list. Matches a string or list of strings and highlights them in the display in Splunk Web. The matching is not case sensitive.

Syntax

highlight <string>...

Required arguments

<string>

Syntax: <string> ...

Description: A space-separated list of strings to highlight in the results. The list you specify is not case-sensitive. Any combination of uppercase and lowercase letters that match the string are highlighted.

Usage

The highlight command is a distributable streaming command. See Command types.
The string that you specify must be a field value. The string cannot be a field name.

You must use the `highlight` command in a search that keeps the raw events and displays output on the Events tab. You cannot use the `highlight` command with commands, such as `stats` which produce calculated or generated results.

**Examples**

**Example 1:**

Highlight the terms "login" and "logout".

```bash
... | highlight login,logout
```

**Example 2:**

Highlight the phrase "Access Denied".

```bash
... | highlight "access denied"
```

**See also**

`rangemap`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `highlight` command.

**history**

**Description**

Use this command to view the search history of the current user. This search history is presented as a set of events or as a table.
Syntax

| history [events=<bool>]

**Required arguments**

None.

**Optional arguments**

**events**

*Syntax:* events=<bool>

*Description:* When you specify `events=true`, the search history is returned as events. This invokes the event-oriented UI which allows for convenient highlighting, or field-inspection. When you specify `events=false`, the search history is returned in a table format for more convenient aggregate viewing.

*Default:* false

Fields returned when `events=false`.

<table>
<thead>
<tr>
<th>Output field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_time</td>
<td>The time that the search was started.</td>
</tr>
<tr>
<td>api_et</td>
<td>The earliest time of the API call, which is the earliest time for which events were requested.</td>
</tr>
<tr>
<td>api_lt</td>
<td>The latest time of the API call, which is the latest time for which events were requested.</td>
</tr>
<tr>
<td>event_count</td>
<td>If the search retrieved or generated events, the count of events returned with the search.</td>
</tr>
<tr>
<td>exec_time</td>
<td>The execution time of the search in integer quantity of seconds into the Unix epoch.</td>
</tr>
<tr>
<td>is_realtime</td>
<td>Indicates whether the search was real-time (1) or historical (0).</td>
</tr>
<tr>
<td>result_count</td>
<td>If the search is a transforming search, the count of results for the search.</td>
</tr>
<tr>
<td>scan_count</td>
<td>The number of events retrieved from a Splunk index at a low level.</td>
</tr>
<tr>
<td>search</td>
<td>The search string.</td>
</tr>
<tr>
<td>Output field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>search_et</td>
<td>The earliest time set for the search to run.</td>
</tr>
<tr>
<td>search_lt</td>
<td>The latest time set for the search to run.</td>
</tr>
<tr>
<td>sid</td>
<td>The search job ID.</td>
</tr>
<tr>
<td>splunk_server</td>
<td>The host name of the machine where the search was run.</td>
</tr>
<tr>
<td>status</td>
<td>The status of the search.</td>
</tr>
<tr>
<td>total_run_time</td>
<td>The total time it took to run the search in seconds.</td>
</tr>
</tbody>
</table>

Usage

The `history` command is a generating command and should be the first command in the search. Generating commands use a leading pipe character.

Examples

**Return search history in a table**

Return a table of the search history. You do not have to specify `events=false`, since that this the default setting.

| history

![Search history in a table](image-url)
Return search history as events

Return the search history as a set of events.

| history events=true

See also

search

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the history command.

iconify

Description

Causes Splunk Web to display an icon for each different value in the list of fields that you specify.

The iconify command adds a field named _icon to each event. This field is the hash value for the event. Within Splunk Web, a different icon for each unique value in the field is displayed in the events list. If multiple fields are listed, the UI displays a different icon for each unique combination of the field values.
Syntax

iconify <field-list>

**Required arguments**

field-list

Syntax: <field>...

Description: Comma or space-delimited list of fields. You cannot specify a wildcard character in the field list.

**Usage**

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

**Examples**

1. **Display a different icon for each eventdata**

   ```
   ... | iconify eventdata
   ```

2. **Display a different icon for unique pairs of field values**

   Display a different icon for unique pair of `clientip` and `method` values.

   ```
   ... | iconify clientip method
   ```

   Here is how Splunk Web displays the results in your **Events List**:

   ![Events List](image)

   **See also**

   `highlight`

392
**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `iconify` command.

**input**

<table>
<thead>
<tr>
<th>This feature is deprecated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <code>input</code> command is deprecated in the Splunk platform as of version 7.1.0. Although this feature continues to function, it might be removed in a future version. To add, enable, or disable inputs, use Splunk Web or edit the <code>inputs.conf</code> file.</td>
</tr>
</tbody>
</table>

In the 7.1.0 Release Notes, see Deprecated features.

**Description**

Adds or disables sources from being processed by the search. Enables or disables inputs in `inputs.conf` with optional sourcetype and index settings. Any additional `attribute=value` are added to `inputs.conf`. If you have Splunk Enterprise, you can view the log of changes to inputs in the following file: `$SPLUNK_HOME/var/log/splunk/inputs.log`.

**Syntax**

```
input (add | remove) [sourcetype=string] [index=string] [string=string]...
```

**Optional arguments**

```
sourcetype

Datatype: <string>
Description: When adding a new input, label the input so the data it acquires uses this sourcetype.
```

```
index

Datatype: <string>
Description: When adding a new input, label the input so the data it acquires is sent to this index. Make sure this index exists.
```

```
string
```
**Datatype:** <string>  
**Description:** Used to specify custom user fields.

**Examples**

**Example 1:**

Remove all csv files that are currently being processed

```
source=*csv | input remove
```

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the input command.

**inputcsv**

**Description**

For Splunk Enterprise deployments, loads search results from the specified .csv file, which is not modified. The filename must refer to a relative path in $SPLUNK_HOME/var/run/splunk/csv (or $SPLUNK_HOME/var/run/splunk/dispatch/<job id>/ if dispatch = true). If the specified file does not exist and the filename does not have an extension, then the Splunk software assumes it has a filename with a .csv extension.

If you run into an issue with the inputcsv command resulting in an error, ensure that your CSV file ends with a BLANK LINE.

**Syntax**

```
| inputcsv [dispatch=<bool>] [append=<bool>] [start=<int>] [max=<int>] [events=<bool>] <filename> [WHERE <search-query>]
```

**Required arguments**

`filename`

**Syntax:** `<filename>`

**Description:** Specify the name of the .csv file, located in $SPLUNK_HOME/var/run/splunk/csv.
Optional arguments

dispatch
Syntax: dispatch=<bool>
Description: When set to true, this argument indicates that the filename is a .csv file in the dispatch directory. The relative path is $SPLUNK_HOME/var/run/splunk/dispatch/<job id>/.
Default: false

append
Syntax: append=<bool>
Description: Specifies whether the data from the .csv file is appended to the current set of results (true) or replaces the current set of results (false).
Default: false

events
Syntax: events=<bool>
Description: Allows the imported results to be treated as events so that a proper timeline and fields picker are displayed.

max
Syntax: max=<int>
Description: Controls the maximum number of events to be read from the file. If max is not specified, there is no limit to the number of events that can be read.
Default: 1000000000 (1 billion)

start
Syntax: start=<int>
Description: Controls the 0-based offset of the first event to be read.
Default: 0

WHERE
Syntax: WHERE <search-query>
Description: Use this clause to improve search performance by prefiltering data returned from the lookup table. Supports a limited set of search query operators: =, !=, <, >, <=, >=, AND, OR, NOT. Any combination of these operators is permitted. Also supports wildcard string searches.
Usage

The `inputcsv` command is an event-generating command. See Command types.

Generating commands use a leading pipe character and should be the first command in a search.

**Appending or replacing results**

If the `append` argument is set to `true`, the Splunk software appends the data from the `.csv` file to the current set of results. The `append` argument is set to `false` by default, which means that it replaces the current result set with the results from the lookup search.

**Working with large CSV lookup tables**

The `WHERE` clause allows you to narrow the scope of the query that `inputlookup` makes against the lookup table. It restricts `inputlookup` to a smaller number of lookup table rows, which can improve search efficiency when you are working with significantly large CSV lookup tables.

**Distributed deployments**

The `inputcsv` command is not compatible with search head pooling and search head clustering.

The command saves the `*.csv` file on the local search head in the `$SPLUNK_HOME/var/run/splunk/` directory. The `*.csv` files are not replicated on the other search heads.

**Examples**

1. **Load results that contain a specific string**

This example loads search results from the `$SPLUNK_HOME/var/run/splunk/csv/all.csv` file. Those that contain the string `error` are saved to the `$SPLUNK_HOME/var/run/splunk/csv/error.csv` file.

```
| inputcsv all.csv | search error | outputcsv errors.csv |
```
2. **Load a specific range of results**

This example loads results 101 to 600 from either the `bar` file, if exists, or from the `bar.csv` file.

```
| inputcsv start=100 max=500 bar
```

3. **Specifying which results to load with operators and expressions**

You can use comparison operators and Boolean expression to specify which results to load. This example loads all of the events from the CSV file `$SPLUNK_HOME/var/run/splunk/csv/students.csv` and then filters out the events that do not match the WHERE clause, where the values in the `age` field are greater than 13, less than 19, but not 16. The search returns a count of the remaining search results.

```
| inputcsv students.csv WHERE (age>=13 age<=19) AND NOT age=16 | stats count
```

4. **Appending multiple CSV files**

You can append the search results of one CSV file to another CSV file by using the `append` command and a subsearch.

```
| inputcsv file1.csv | append [| inputcsv file2.csv]
```

**See also**

`outputcsv`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `inputcsv` command.

**inputintelligence**

The `inputintelligence` command is used with Splunk Enterprise Security.

For information about this command, see Use generic intelligence in search with `inputintelligence` in *Administer Splunk Enterprise Security*. 
inputlookup

Description

Use the `inputlookup` command to search the contents of a lookup table. The lookup table can be a CSV lookup or a KV store lookup.

Syntax

```
| inputlookup [append=<bool>] [start=<int>] [max=<int>] [<filename> | <tablename>] [WHERE <search-query>]
```

**Required arguments**

<filename>

  **Syntax:** `<string>`
  **Description:** The name of the lookup file must end with `.csv` or `.csv.gz`. If the lookup does not exist, a warning message is displayed (but no syntax error is generated).

<tablename>

  **Syntax:** `<string>`
  **Description:** The name of the lookup table as specified by a stanza name in the `transforms.conf` file. The lookup table can be configured for any lookup type (CSV, external, or KV store).

**Optional arguments**

append

  **Syntax:** `append=<bool>`
  **Description:** If set to `true`, the data returned from the lookup file is appended to the current set of results rather than replacing it. Defaults to `false`.

max

  **Syntax:** `max=<int>`
  **Description:** Specify the maximum number of events to be read from the file. Defaults to `1000000000`.

start

  **Syntax:** `start=<int>`
**Description:** Specify the 0-based offset of the first event to read. If `start=0`, it begins with the first event. If `start=4`, it begins with the fifth event. Defaults to 0.

WHERE clause

**Syntax:** WHERE <search-query>

**Description:** Use this clause to improve search performance by prefiltering data returned from the lookup table. Supports a limited set of search query operators: `=, !=, <, >, <=, >=, AND, OR, NOT. Any combination of these operators is permitted. Also supports wildcard string searches.

**Usage**

The `inputlookup` command is an event-generating command. See Command types.

Generating commands use a leading pipe character and should be the first command in a search.

The lookup can be a file name that ends with `.csv` or `.csv.gz`, or a lookup table configuration in the `transforms.conf` file.

**Appending or replacing results**

If `append=true`, data from the lookup file or KV store collection is appended to the current set of results. By default, `append=false` which means that the current result set is replaced with the results from the lookup search.

**Working with large CSV lookup tables**

The WHERE clause allows you to narrow the scope of the query that `inputlookup` makes against the lookup table. It restricts `inputlookup` to a smaller number of lookup table rows, which can improve search efficiency when you are working with significantly large lookup tables.

**Testing geometric lookup files**

You can use the `inputlookup` command to verify that the geometric features on the map are correct. The syntax is `| inputlookup <your_lookup>`.

1. For example, to verify that the geometric features in built-in `geo_us_states` lookup appear correctly on the choropleth map, run the following search:
| inputlookup geo_us_states

2. On the Visualizations tab, zoom in to see the geometric features. In this example, the states in the United States.

Additional information

For more information about creating lookups, see About lookups in the Knowledge Manager Manual.

For more information about the App Key Value store, see About KV store in the Admin Manual.

Examples

Example 1: Read in a usertogroup lookup table that is defined in transforms.conf.

| inputlookup usertogroup

Example 2: Read in a usertogroup table that is defined by a stanza in transforms.conf. Append the fields to any current results.

| inputlookup append=t usertogroup

Example 3: Search the users.csv lookup file (under $SPLUNK_HOME/etc/system/lookups or $SPLUNK_HOME/etc/apps/<app_name>/lookups).

| inputlookup users.csv

Example 4: Search the contents of the KV store collection kvstorecoll that have a CustID value greater than 500 and a CustName value that begins with the letter P. The collection is referenced in a lookup table called kvstorecoll_lookup. Provide a count of the events received from the table.

| inputlookup kvstorecoll_lookup where (CustID>500) AND (CustName="P*") | stats count

Example 5: View internal key ID values for the KV store collection kvstorecoll, using the lookup table kvstorecoll_lookup. The internal key ID is a unique identifier for each record in the collection. This requires usage of the eval and table commands.
Example 6: Update field values for a single KV store collection record. This requires usage of `inputlookup`, `outputlookup`, and `eval`. The record is indicated by the its internal key ID (the `_key` field) and this search updates the record with a new customer name and customer city. The record belongs to the KV store collection `kvstorecoll`, which is accessed through the lookup table `kvstorecoll_lookup`.

```
| inputlookup kvstorecoll_lookup | search _key=544948df3ec32d7a4c1d9755 | eval CustName="Marge Simpson" | eval CustCity="Springfield" | outputlookup kvstorecoll_lookup append=True key_field=_key
```

Example 7: Write the contents of a CSV file to the KV store collection `kvstorecoll` using the lookup table `kvstorecoll_lookup`. This requires usage of both `inputlookup` and `outputlookup`.

```
| inputlookup customers.csv | outputlookup kvstorecoll_lookup
```

See also

`inputcsv`, `join`, `lookup`, `outputlookup`

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `inputlookup` command.

iplocation

Description

Extracts location information from IP addresses by using 3rd-party databases. This command supports IPv4 and IPv6.

The IP address that you specify in the `ip-address-fieldname` argument, is looked up in the database. Fields from that database that contain location information are added to each event. The setting used for the `allfields` argument determines which fields are added to the events.
Because all the information might not be available for each IP address, an event can have empty field values.

For IP addresses which do not have a location, such as internal addresses, no fields are added.

**Syntax**

```
iplocation [prefix=<string>] [allfields=<bool>] [lang=<string>]
<ip-address-fieldname>
```

**Required arguments**

- **ip-address-fieldname**
  
  **Syntax:** `<field>`
  
  **Description:** Specify an IP address field, such as `clientip`.

**Optional arguments**

- **allfields**
  
  **Syntax:** `allfields=<bool>`
  
  **Description:** Specifies whether to add all of the fields from the database to the events. If set to `true`, adds the fields City, Continent, Country, lat (latitude), lon (longitude), MetroCode, Region, and Timezone.
  
  **Default:** false. Only the City, Country, lat, lon, and Region fields are added to the events.

- **lang**
  
  **Syntax:** `lang=<string>`
  
  **Description:** Render the resulting strings in different languages. For example, use "lang=es" for Spanish. The set of languages depends on the geoiip database that is used. To specify more than one language, separate them with a comma. This also indicates the priority in descending order. Specify "lang=code" to return the fields as two letter ISO abbreviations.

- **prefix**
  
  **Syntax:** `prefix=<string>`
  
  **Description:** Specify a string to prefix the field name. With this argument you can add a prefix to the added field names to avoid name collisions with existing fields. For example, if you specify `prefix=iploc_ the field names that are added to the events become iploc_City, iploc_County, iploc_lat, and so forth.
  
  **Default:** NULL/empty string
Usage

The iplocation command is a distributable streaming command. See Command types.

The Splunk software ships with a copy of the GeoLite2-City.mmmdb database file. This file is located in the $SPLUNK_HOME/share/ directory.

Updating the MMDB file

You can replace the version of the .mmdb file that ships with the Splunk software with a copy of the paid version of the file or with a monthly update of the free version of the file.

2. Copy the file to the search head on your Splunk Enterprise instance.
3. Expand the GZ file.
4. Stop any real-time searches that are running.
5. Copy the GeoLite2-City.mmmdb file to the $SPLUNK_HOME/share/ directory to overwrite the file there.
6. Restart the real-time searches.

Impact of upgrading Splunk software

When you upgrade your Splunk platform, the GeoLite2-City.mmmdb file in the share directory is replaced by the version of the file that ships with the Splunk software. One option is to store the MMDB file in a different path.

Storing the MMDB file in a different path

If you prefer to update the GeoLite2-City.mmmdb file yourself, for example if you use a paid version of the file, you can store the MMDB file in a different path. The path that is used by the Splunk software to access the file must be updated.

Prerequisites

- Only users with file system access, such as system administrators, can specify a different path to the MMDB file in the limits.conf file.
- Review the steps in How to edit a configuration file in the Admin Manual.
- You can have configuration files with the same name in your default, local, and app directories. Read Where you can place (or find) your modified
configuration files in the *Admin Manual*.

Never change or copy the configuration files in the default directory. The files in the default directory must remain intact and in their original location. Make the changes in the local directory.

If you are using Splunk Cloud, updates to the MMDB file are provided ONLY via Splunk version upgrades. If you wish to discuss or request this, please file a Support ticket.

**Steps**

1. Open the local *limits.conf* file for the Search app. For example, $SPLUNK_HOME/etc/system/local.
2. Add the *[iplocation]* stanza.
3. Add the *db_path* setting and specify the absolute path to the *GeoLite2-City.mmdb* file. The *db_path* setting does not support standard Splunk environment variables such as $SPLUNK_HOME.
   For example: 
   
   ```
   db_path = /Applications/Splunk/mmdb/GeoLite2-City.mmdb
   ```
   specifies a new directory called mmdb.
4. Ensure a copy of the MMDB file is stored in the
   ```
   ../Applications/Splunk/mmdb/
   ```
5. Because you are editing the path to the MMDB file, you should restart the Splunk server.

**Storing the MMDB file with a different name**

Alternatively, you can add the updated MMDB to the share directory using a different name and then specify that name in the *db_path* setting. For example:

```
db_path = /Applications/Splunk/share/GeoLite2-City_paid.mmdb
```

**The MMDB file and distributed deployments**

The *iplocation* command is a distributable **streaming command**, which means that it can be processed on the indexers. The share directory is not part of the knowledge bundle. If you update the MMDB file in the share directory, the updated file is not automatically sent to the indexers in a distributed deployment. To add the MMDB file to the indexers, use the tools that you typically use to push files to the indexers.
Examples

1. Add location information to web access events

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

Add location information to web access events. By default, the `iplocation` command adds the City, Country, lat, lon, and Region fields to the results.

`sourcetype=access_* | iplocation clientip`

2. Search for client errors and return the first 20 results

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

Search for client errors in web access events, returning only the first 20 results. Add location information and return a table with the IP address, City, and Country for each client error.

`sourcetype=access_* status>=400 | head 20 | iplocation clientip | table clientip, status, City, Country`

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>clientip</th>
<th>status</th>
<th>City</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>182.236.164.11</td>
<td>408</td>
<td>Zhengzhou</td>
<td>China</td>
</tr>
<tr>
<td>198.35.1.75</td>
<td>500</td>
<td>Princeton</td>
<td>United States</td>
</tr>
<tr>
<td>198.35.1.75</td>
<td>404</td>
<td>Princeton</td>
<td>United States</td>
</tr>
<tr>
<td>198.35.1.75</td>
<td>406</td>
<td>Princeton</td>
<td>United States</td>
</tr>
<tr>
<td>198.35.1.75</td>
<td>500</td>
<td>Princeton</td>
<td>United States</td>
</tr>
<tr>
<td>221.204.246.72</td>
<td>503</td>
<td>Taiyuan</td>
<td>China</td>
</tr>
<tr>
<td>1.192.86.205</td>
<td>503</td>
<td>Amesbury</td>
<td>United States</td>
</tr>
<tr>
<td>91.205.189.15</td>
<td>406</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 3. Add a prefix to the fields added by the iplocation command

Prefix the fields added by the `iplocation` command with `iploc_`. Add all of the fields in the `GeoLite2-City.mmdb` database file to the results.

```
sourcetype = access_* | iplocation prefix=iploc_ allfields=true clientip | fields iploc_*
```

### 4. Generate a choropleth map using IP addresses

Generate a choropleth map of your data like the one below using the `iplocation` command. See Use IP addresses to generate a choropleth map in Dashboards and Visualizations.

---

### Table

<table>
<thead>
<tr>
<th>clientip</th>
<th>status</th>
<th>City</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>216.221.226.11</td>
<td>505</td>
<td>Redwood City</td>
<td>United States</td>
</tr>
<tr>
<td>216.221.226.11</td>
<td>404</td>
<td>Redwood City</td>
<td>United States</td>
</tr>
<tr>
<td>195.2.240.99</td>
<td>400</td>
<td></td>
<td>Russa</td>
</tr>
</tbody>
</table>
**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the iplocation command.

**join**

**Description**

Use the `join` command to combine the results of a subsearch with the results of a main search. One or more of the fields must be common to each result set. You can also combine a search result set to itself using the `selfjoin` command.

If you are familiar with SQL but new to SPL, see Splunk SPL for SQL users.

**Alternative commands**

For flexibility and performance, consider using one of the following commands if you do not require join semantics. These commands provide event grouping and correlations using time and geographic location, transactions, subsearches, field lookups, and joins.

<table>
<thead>
<tr>
<th>Command</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>append</code></td>
<td>To append the results of a subsearch to the results of your current search. The events from both result sets are retained.</td>
</tr>
<tr>
<td></td>
<td>• Use only with historical data. The <code>append</code> command does not produce correct results if used in a real-time search.</td>
</tr>
<tr>
<td></td>
<td>• If you use <code>append</code> to combine the events, use a <code>stats</code> command to group the events in a meaningful way. You cannot use a <code>transaction</code> command after you use an <code>append</code> command.</td>
</tr>
<tr>
<td><code>appendcols</code></td>
<td>Appends the fields of the <code>subsearch</code> results with the input search result fields. The first subsearch result is merged with the first main result, the second subsearch result is merged with the second main result, and so on.</td>
</tr>
<tr>
<td><code>lookup</code></td>
<td>Use when one of the result sets or source files remains static or rarely changes. For example, a file from an external system such as a CSV file.</td>
</tr>
<tr>
<td><strong>Command</strong></td>
<td><strong>Use</strong></td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>search</strong></td>
<td>In the most simple scenarios, you might need to search only for sources using the OR operator and then use a <strong>stats</strong> or <strong>transaction</strong> command to perform the grouping operation on the events.</td>
</tr>
</tbody>
</table>
| **stats**   | To group events by a field and perform a statistical function on the events. For example to determine the average duration of events by host name.  
- To use **stats**, the field must have a unique identifier.  
- To view the raw event data, use the **transaction** command instead. |
| **transaction** | Use **transaction** in the following situations.  
- To group events by using the **eval** command with a conditional expression, such as **if**, **case**, or **match**.  
- To group events by using a recycled field value, such as an ID or IP address.  
- To group events by using a pattern, such as a start or end time for the event.  
- To break up groups larger than a certain duration. For example, when a transaction does not explicitly end with a message and you want to specify a maximum span of time after the start of the transaction.  
- To display the raw event data for the grouped events. |

For information about when to use a join, see the flowchart in About event grouping and correlation in the Search Manual.

**Syntax**

```
join [join-options...] [field-list] subsearch
```

**Required arguments**

**subsearch**

**Syntax:** "[ subsearch "]"

**Description:** A secondary search where you specify the source of the events that you want to join to. The subsearch must be enclosed in square brackets. The results of the subsearch should not exceed available
Limitations on the subsearch for the `join` command are specified in the `limits.conf.spec` file. The limitations include the maximum subsearch to join against, the maximum search time for the subsearch, and the maximum time to wait for subsearch to fully finish. See Subsearches in the Search Manual.

**Optional arguments**

**field-list**

Syntax: `<field>, <field>, ...`

Description: Specify the fields to use for the join. If no fields are specified, all of the fields that are common to both result sets are used.

Field names must match, not just in name but also in case. You cannot join `product_id` with `product_ID`. You must first change the case of the field in the subsearch to match the field in the main search.

**join-options**

Syntax: `type=(inner | outer | left) | usetime=<bool> | earlier=<bool> | overwrite=<bool> | max=<int>`

Description: Options to the join command. Use either `outer` or `left` to specify a left outer join.

**Descriptions for the join-options argument**

**type**

Syntax: `type=inner | outer | left`

Description: Indicates the type of join to perform. The difference between an `inner` and a `left` (or `outer`) join is how the events are treated in the main search that do not match any of the events in the subsearch. In both `inner` and `left` joins, events that match are joined. The results of an `inner` join do not include events from the main search that have no matches in the subsearch. The results of a `left` (or `outer`) join includes all of the events in the main search and only those values in the subsearch have matching field values.

Default: `inner`

**usetime**

Syntax: `usetime=<bool>`

Description: A Boolean value that Indicates whether to use time to limit the matches in the subsearch results. Used with the `earlier` option to limit
the subsearch results to matches that are earlier or later than the main search results.

**Default:** true

earlier

**Syntax:** earlier=<bool>

**Description:** If `usetime=true` and `earlier=true`, the main search results are matched only against earlier results from the subsearch. If `earlier=false`, the main search results are matched only against later results from the subsearch. Results that occur at the same time (second) are not eliminated by either value.

**Default:** true

overwrite

**Syntax:** overwrite=<bool>

**Description:** Indicates whether fields from the subresults overwrite the fields from the main results, if the fields have the same field name.

**Default:** true

max

**Syntax:** max=<int>

**Description:** Specifies the maximum number of subsearch results that each main search result can join with. If set to `max=0`, there is no limit.

**Default:** 1

Usage

The `join` command is a centralized streaming command when there is a defined set of fields to join to. See Command types.

Use the `join` command when the results of the subsearch are relatively small, for example 50,000 rows or less. To minimize the impact of this command on performance and resource consumption, Splunk software imposes some default limitations on the subsearch. See the subsearch section in the syntax for more information about these limitations.

**One-to-many and many-to-many relationships**

To return matches for one-to-many, many-to-one, or many-to-many relationships, include the `max` argument in your `join` syntax and set the value to 0. By default `max=1`, which means that the subsearch returns only the first result from the subsearch. Setting the value to a higher number or to 0, which is unlimited, returns multiple results from the subsearch.
Examples

Example 1

Combine the results from a main search with the results from a subsearch search vendors. The result sets are joined on the product_id field, which is common to both sources.

... | join product_id [search vendors]

Example 2

If the field names in the sources do not match, you can rename the field in the subsearch result set. The field in the main search is product_id. The field in the subsearch is pid.

Note: The field names must match in name and in case. You cannot join product_id with product_ID.

... | join product_id [search vendors | rename pid AS product_id]

Example 3

By default, only the first row of the subsearch that matches a row of the main search is returned. To return all of the matching subsearch rows, include the max=<int> argument and set the value to 0. This argument joins each matching subsearch row with the corresponding main search row.

... | join product_id max=0 [search vendors]

Example 4

The dashboards and alerts in the distributed management console shows you performance information about your Splunk deployment. The Resource Usage: Instance dashboard contains a table that shows the machine, number of cores, physical memory capacity, operating system, and CPU architecture.

To display the information in the table, use the following search. This search includes a join command. The search uses the information in the dmc_assets table to look up the instance name and machine name. The search then uses the serverName field to join the information with information from the /services/server/info REST endpoint. The /services/server/info is the URI path to the Splunk REST API endpoint that provides hardware and operating
system information for the machine. The $splunk_server$ part of the search is a
token variable.

| inputlookup dmc_assets |
| search serverName = $splunk_server$ |
| stats first(serverName) AS serverName, first(host) AS host, first(machine) AS machine |
| join type=left serverName |
| [ | rest splunk_server=$splunk_server$ /services/server/info |
| fields serverName, numberOfCores, physicalMemoryMB, os_name, cpu_arch |
| fields machine numberOfCores physicalMemoryMB os_name cpu_arch |
| rename machine AS Machine, numberOfCores AS "Number of Cores", |
| physicalMemoryMB AS "Physical Memory Capacity (MB)", os_name AS |
| "Operating System", |
| cpu_arch AS "CPU Architecture" |

See also

selfjoin, append, set, appendcols

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the join command.

kmeans

Description

Partitions the events into k clusters, with each cluster defined by its mean value. Each event belongs to the cluster with the nearest mean value. Performs k-means clustering on the list of fields that you specify. If no fields are specified, performs the clustering on all numeric fields. Events in the same cluster are moved next to each other. You have the option to display the cluster number for each event.

Syntax

kmeans [kmeans-options...] [field-list]
**Required arguments**

None.

**Optional arguments**

**field-list**

**Syntax:** `<field> ...`

**Description:** Specify a space separated list of the exact fields to use for the join.

**Default:** If no fields are specified, uses all numerical fields that are common to both result sets. Skips events with non-numerical fields.

**kmeans-options**

**Syntax:** `<reps> | <iters> | <t> | <k> | <cnumfield> | <distype> | <showcentroid>`

**Description:** Options for the `kmeans` command.

**kmeans options**

**reps**

**Syntax:** `reps=<int>`

**Description:** Specify the number of times to repeat kmeans using random starting clusters.

**Default:** 10

**iters**

**Syntax:** `maxiters=<int>`

**Description:** Specify the maximum number of iterations allowed before failing to converge.

**Default:** 10000

**t**

**Syntax:** `t=<num>`

**Description:** Specify the algorithm convergence tolerance.

**Default:** 0

**k**

**Syntax:** `k=<int> | <int>-<int>`

**Description:** Specify as a scalar integer value or a range of integers. When provided as single number, selects the number of clusters to use. This produces events annotated by the cluster label. When expressed as a range, clustering is done for each of the cluster counts in the range and
a summary of the results is produced. These results express the size of the clusters, and a 'distortion' field which represents how well the data fits those selected clusters. Values must be greater than 1 and less than maxkvalue (see Limits section).

Default: k=2

cnumfield

Syntax: cfield=<field>

Description: Names the field to annotate the results with the cluster number for each event.

Default: CLUSTERNUM

distype

Syntax: dt= ( l1 | l1norm | cityblock | cb ) | ( l2 | l2norm | sq | sqeuclidean )
| ( cos | cosine )

Description: Specify the distance metric to use. The l1, l1norm, and cb distance metrics are synonyms for cityblock. The l2, l2norm, and sq distance metrics are synonyms for sqeuclidean or sqEuclidean. The cos distance metric is a synonym for cosine.

Default: sqeuclidean

showcentroid

Syntax: showcentroid= true | false

Description: Specify whether to expose the centroid centers in the search results (showcentroid=true) or not.

Default: true

Usage

Limits

The number of clusters to collect the values into -- k -- is not permitted to exceed maxkvalue. The maxkvalue is specified in the limits.conf file, in the [kmeans] stanza. The maxkvalue default is 1000.

When a range is given for the k option, the total distance between the beginning and ending cluster counts is not permitted to exceed maxkrange. The maxkrange is specified in the limits.conf file, in the [kmeans] stanza. The maxkrange default is 100.

The above limits are designed to avoid the computation work becoming unreasonably expensive.
The total number of values which are clustered by the algorithm (typically the number of input results) is limited by the `maxdatapoints` parameter in the `[kmeans]` stanza of `limits.conf`. If this limit is exceeded at runtime, a warning message displays in Splunk Web. This defaults to 100000000 or 100 million. This `maxdatapoints` limit is designed to avoid exhausting memory.

**Examples**

**Example 1:** Group search results into 4 clusters based on the values of the "date_hour" and "date_minute" fields.

```bash
... | kmeans k=4 date_hour date_minute
```

**Example 2:** Group results into 2 clusters based on the values of all numerical fields.

```bash
... | kmeans
```

**See also**

`anomalies`, `anomalousvalue`, `cluster`, `outlier`,

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `kmeans` command.

**kvform**

**Description**

Extracts key/value pairs from events based on a form template that describes how to extract the values.

**Syntax**

```bash
kvform [form=<string>] [field=<field>]
```
Optional arguments

form
Syntax: form=<string>
Description: Specify a .form file located in a
$SPLUNK_HOME/etc/apps/*/forms/ directory.

field
Syntax: field=<field_name>
Description: Uses the field name to look for .form files that correspond to
the field values for that field name. For example, your Splunk deployment
uses the splunkd and mongod sourcetypes. If you specify
field=sourcetype, the kvform command looks for the splunkd.form and
mongod.form in the $SPLUNK_HOME/etc/apps/*/forms/ directory.
Default: sourcetype

Usage

Before you can use the kvform command, you must:

• Create the forms directory in the appropriate application path. For
  example $SPLUNK_HOME/etc/apps/<app_name>/forms.
• Create the .form files and add the files to the forms directory.

If you have Splunk Cloud and want to install form files, file a Support ticket.

Format for the .form files

A .form file is essentially a text file of all static parts of a form. It might be
interspersed with named references to regular expressions of the type found in
the transforms.conf file.

An example .form file might look like this:

Students Name: [[string:student_name]]
Age: [[int:age]] Zip: [[int:zip]]

Specifying a form

If the form argument is specified, the kvform command uses the
<form_name>.form file found in the Splunk configuration forms directory. For
example, if form=sales_order, the kvform command looks for a
sales_order.form file in the $SPLUNK_HOME/etc/apps/<app_name>/forms directory for all apps. All the events processed are matched against the form, trying to extract values.

**Specifying a field**

If you specify the field argument, the kvform command looks for forms in the forms directory that correspond to the values for that field. For example, if you specify *field=error_code*, and an event has the field value *error_code=404*, the command looks for a form called *404.form* in the $SPLUNK_HOME/etc/apps/<app_name>/forms directory.

**Default value**

If no form or field argument is specified, the kvform command uses the default value for the field argument, which is *sourcetype*. The kvform command looks for <sourcetype_value>.form files to extract values.

**Examples**

1. Extract values using a specific form

Use a specific form to extract values from.

```
... | kvform form=sales_order
```

2. Extract values using a field name

Specify *field=sourcetype* to extract values from forms such as splunkd.form and mongod.form. If there is a form for a source type, values are extracted from that form. If one of the source types is access_combined but there is no access_combined.form file, that source type is ignored.

```
... | kvform field=sourcetype
```

3. Extract values using the eventtype field

```
... | kvform field=eventtype
```

**See also**

extract, multikv, rex, xmlkv
Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the kvform command.

loadjob

Description

Loads events or results of a previously completed search job. The artifacts to load are identified either by the search job id or a scheduled search name and the time range of the current search. If a savedsearch name is provided and multiple artifacts are found within that range the latest artifacts are loaded.

A search head cluster can run the loadjob command only on scheduled saved searches. A search head cluster runs searches on results or artifacts that the search head cluster replicates. You cannot run the loadjob command on ad hoc or real-time searches. For more information on artifact replication, see "Search head clustering architecture" in the Distributed Search manual.

Syntax

| loadjob (<sid> | <savedsearch>) [result-event] [delegate] [artifact_offset] [ignore_running]

Required arguments

sid

Syntax: <string>
Description: The search ID of the job whose artifacts need to be loaded, for example: 1233886270.2

savedsearch

Syntax: savedsearch="<user-string>:<app-string>:<search-name-string>"
Description: The unique identifier of a saved search whose artifacts need to be loaded. A saved search is uniquely identified by the triplet {user, app, savedsearch name}, for example: savedsearch="admin:search:my Saved Search" There is no method to specify a wildcard or match-all behavior. All portions of the triplet must be provided.
Optional arguments

result-event
   Syntax: events=<bool>
   Description: events=true loads events, while events=false loads results.
   Defaults: false

delegate
   Syntax: job_delegate=<string>
   Description: When specifying a savedsearch, this option selects jobs that were started by the given user. Scheduled jobs will be run by the delegate "scheduler". Dashboard-embedded searches will be run in accordance with the savedsearch’s dispatchAs parameter (typically the owner of the search).
   Defaults: scheduler

artifact_offset
   Syntax: artifact_offset=<int>
   Description: Selects a search artifact other than the most recent matching one. For example, if artifact_offset=1, the second most recent artifact will be used. If artifact_offset=2, the third most recent artifact will be used. If artifact_offset=0, selects the most recent. A value that selects past all available artifacts will result in an error.
   Default: 0

ignore_running
   Syntax: ignore_running=<bool>
   Description: Skip over artifacts whose search is still running.
   Default: true

Usage

The loadjob command is an event-generating command. See Command types.

Generating commands use a leading pipe character and should be the first command in a search.

The loadjob command can be used for a variety of purposes, but one of the most useful is to run a fairly expensive search that calculates statistics. You can use loadjob searches to display those statistics for further aggregation, categorization, field selection and other manipulations for charting and display.
After a search job has completed and the results are cached, you can use this command to access or load the results.

**Examples**

**Example 1:** Loads the results of the latest scheduled execution of savedsearch MySavedSearch in the 'search' application owned by admin

```
| loadjob savedsearch="admin:search:MySavedSearch"
```

**Example 2:** Loads the events that were generated by the search job with id=1233886270.2

```
| loadjob 1233886270.2 events=true
```

**See also**

inputcsv, savedsearch

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the loadjob command.

**localize**

**Description**

The `localize` command generates results that represent a list of time contiguous event regions. An event region is a period of time in which consecutive events are separated, at most, by the `maxpause` time value. The regions found can be expanded using the `timeafter` and `timebefore` arguments.

The regions discovered by the `localize` command are meant to be fed into the `map` command. The `map` command uses a different region for each iteration.

**Syntax**

```
localize [maxpause] [timeafter] [timebefore]
```
Optional arguments

maxpause
   Syntax: maxpause=<int>(s|m|h|d)
   Description: Specify the maximum (inclusive) time between two consecutive events in a contiguous time region.
   Default: 1m

timeafter
   Syntax: timeafter=<int>(s|m|h|d)
   Description: Specify the amount of time to add to the output endtime field (expand the time region forward in time).
   Default: 30s

timebefore
   Syntax: timebefore=<int>(s|m|h|d)
   Description: Specify the amount of time to subtract from the output starttime field (expand the time region backwards in time).
   Default: 30s

Usage

Expanding event ranges

You can expand the event range after the last event or before the first event in the region. These expansions are done arbitrarily, possibly causing overlaps in the regions if the values are larger than maxpause.

Event region order

The regions are returned in search order, or descending time for historical searches and data-arrival order for realtime search. The time of each region is the initial pre-expanded start-time.

Other information returned by the localize command

The localize command also reports:

- The number of events in the range
- The range duration in seconds
- The region density defined as the number of events in range divided by <range duration> - events per second.
Examples

1. **Search the time range of each previous result for the term “failure”**

   
   
   ... | localize maxpause=5m | map search="search failure
   starttimeu=$starttime$ endtimeu=$endtime$"

2: **Finds suitable regions around where “error” occurs**

   Searching for “error” and calling the `localize` command finds suitable regions around where error occurs and passes each on to the search inside of the `map` command. Each iteration works with a specific time range to find potential transactions.

   
   error | localize | map search="search starttimeu::$starttime$
   endtimeu::$endtime$ | transaction uid,qid maxspan=1h"

See also

map, transaction

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `localize` command.

localop

Description

Prevents subsequent commands from being executed on remote peers. Tells the search to run subsequent commands locally, instead.

The `localop` command forces subsequent commands to be part of the reduce step of the `mapreduce` process.

Syntax

`localop`
Examples

Example 1:

The iplocation command in this case will never be run on remote peers. All events from remote peers that originate from the initial search, which was for the terms FOO and BAR, are forwarded to the search head. The search head is where the iplocation command is run.

FOO BAR | localop | iplocation clientip

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the localop command.

lookup

Description

Use the lookup command to invoke field value lookups.

For information about the types of lookups you can define, see About lookups in the Knowledge Manager Manual.

Syntax

lookup [local=<bool>] [update=<bool>] <lookup-table-name> ( <lookup-field> [AS <event-field>] )... [ OUTPUT | OUTPUTNEW (<lookup-destfield> [AS <event-destfield>] )... ]

Note: The lookup command can accept multiple lookup and event fields and destfields. For example:

...| lookup <lookup-table-name> <lookup-field1> AS <event-field1>, <lookup-field2> AS <event-field2> OUTPUTNEW <lookup-destfield1> AS <event-destfield1>, <lookup-destfield2> AS <event-destfield2>
**Required arguments**

<lookup-table-name>

**Syntax:** <string>

**Description:** Refers to a stanza name in the transforms.conf file. This stanza specifies the location of the lookup table file.

**Optional arguments**

`local`

**Syntax:** local=<bool>

**Description:** If local=true, forces the lookup to run on the search head and not on any remote peers.

**Default:** false

`update`

**Syntax:** update=<bool>

**Description:** If the lookup table is modified on disk while the search is running, real-time searches do not automatically reflect the update. To do this, specify update=true. This does not apply to searches that are not real-time searches. This implies that local=true.

**Default:** false

<lookup-field>

**Syntax:** <string>

**Description:** Refers to a field in the lookup table to match against the events. You can specify multiple <lookup-field> values.

<event-field>

**Syntax:** <string>

**Description:** Refers to a field the events from which to acquire the value to match in the lookup table. You can specify multiple <event-field> values.

**Default:** The value of the <lookup-field>.

<lookup-destfield>

**Syntax:** <string>

**Description:** Refers to a field in the lookup table to be copied into the events. You can specify multiple <lookup-destfield> values.

<event-destfield>

**Syntax:** <string>
**Description:** A field in the events. You can specify multiple `<event-destfield>` values.

**Default:** The value of the `<lookup-destfield>` argument.

### Usage

The `lookup` command is a distributable streaming command when `local=false`, which is the default setting. See Command types.

When using the `lookup` command, if an OUTPUT or OUTPUTNEW clause is not specified, all of the fields in the lookup table that are not the match field are used as output fields. If the OUTPUT clause is specified, the output lookup fields overwrite existing fields. If the OUTPUTNEW clause is specified, the lookup is not performed for events in which the output fields already exist.

**Optimizing your lookup search**

If you are using the `lookup` command in the same pipeline as a transforming command, and it is possible to retain the field you will lookup on after the transforming command, do the lookup after the transforming command. For example, run:

```bash
sourcetype=access_* | stats count by status | lookup status_desc status OUTPUT description
```

and not:

```bash
sourcetype=access_* | lookup status_desc status OUTPUTNEW description | stats count by description
```

The lookup in the first search is faster because it only needs to match the results of the stats command and not all the Web access events.

### Basic example

1. **Lookup users and return the corresponding group the user belongs to**

Suppose you have a lookup table specified in a stanza named `usertogroup` in the `transforms.conf` file. This lookup table contains (at least) two fields, `user` and `group`. Your events contain a field called `local_user`. For each event, the following search checks to see if the value in the field `local_user` has a corresponding value in the `user` field in the lookup table. For any entries that
match, the value of the `group` field in the lookup table is written to the field `user_group` in the event.

```plaintext
... | lookup usertogroup user as local_user OUTPUT group as user_group
```

### Extended example

#### 2. Lookup price and vendor information and return the count for each product sold by a vendor

This example uses the tutorialdata.zip file from the Search Tutorial. You can download this file and follow the instructions to upload the tutorial data into your Splunk deployment. Additionally, this example uses the prices.csv and the vendors.csv files. To follow along with this example in your Splunk deployment, download these CSV files and complete the steps in the Use field lookups section of the Search Tutorial for both the `prices.csv` and the `vendors.csv` files. When you create the lookup definition for the `vendors.csv` file, name the lookup `vendors_lookup`. You can skip the step in the tutorial that makes the lookups automatic.

This example calculates the count of each product sold by each vendor.

The `prices.csv` file contains the product names, price, and code. For example:

<table>
<thead>
<tr>
<th>productId</th>
<th>product_name</th>
<th>price</th>
<th>sale_price</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB-SG-G01</td>
<td>Mediocre Kingdoms</td>
<td>24.99</td>
<td>19.99</td>
<td>A</td>
</tr>
<tr>
<td>DC-SG-G02</td>
<td>Dream Crusher</td>
<td>39.99</td>
<td>24.99</td>
<td>B</td>
</tr>
<tr>
<td>FS-SG-G03</td>
<td>Final Sequel</td>
<td>24.99</td>
<td>16.99</td>
<td>C</td>
</tr>
<tr>
<td>WC-SH-G04</td>
<td>World of Cheese</td>
<td>24.99</td>
<td>19.99</td>
<td>D</td>
</tr>
</tbody>
</table>

The `vendors.csv` file contains vendor information, such as vendor name, city, and ID. For example:

<table>
<thead>
<tr>
<th>Vendor</th>
<th>VendorCity</th>
<th>VendorID</th>
<th>VendorLatitude</th>
<th>VendorLongitude</th>
<th>VendorStateProvince</th>
<th>VendorCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anchorage Gaming</td>
<td>Anchorage</td>
<td>1001</td>
<td>61.17440033</td>
<td>-149.9960022</td>
<td>Alaska</td>
<td>United States</td>
</tr>
<tr>
<td>Games of Salt Lake</td>
<td>Salt Lake City</td>
<td>1002</td>
<td>40.78839874</td>
<td>-111.9779968</td>
<td>Utah</td>
<td>United States</td>
</tr>
<tr>
<td>New Jack Games</td>
<td>New York</td>
<td>1003</td>
<td>40.63980103</td>
<td>-73.77890015</td>
<td>New York</td>
<td>United States</td>
</tr>
</tbody>
</table>
The search will query the `vendor_sales.log` file, which is part of the `tutorialdata.zip` file. The `vendor_sales.log` file contains the VendorID, Code, and AcctID fields. For example:

<table>
<thead>
<tr>
<th>Date</th>
<th>Event Log</th>
<th>VendorID</th>
<th>Code</th>
<th>AcctID</th>
</tr>
</thead>
<tbody>
<tr>
<td>13/Mar/2018:18:24:02</td>
<td>VendorID=5036 Code=B AcctID=6024298300471575</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following search calculates the count of each product sold by each vendor and uses the time range All time.

```
sourcetype=vendor_* | stats count by Code VendorID | lookup prices_lookup Code OUTPUTNEW product_name
```

- The `stats` command calculates the count by Code and VendorID.
- The `lookup` command uses the `prices_lookup` to match the Code field in each event and return the product names.

The search results are displayed on the Statistics tab.
You can extend the search to display more information about the vendor by using the vendors_lookup.

Use the `table` command to return only the fields that you need. In this example you want the `product_name`, `VendorID`, and `count` fields. Use the `vendors_lookup` file to output all the fields in the `vendors.csv` file that match the `VendorID` in each event.

```
sourcetype=vendor_* | stats count by Code VendorID | lookup prices_lookup Code OUTPUTNEW product_name | table product_name VendorID count | lookup vendors_lookup VendorID
```

The revised search results are displayed on the Statistics tab.

To expand the search to display the results on a map, see the geostats command.

See also

Commands:

- `appendcols`
- `inputlookup`
- `outputlookup`

Related topics:
About lookups in the Knowledge Manager Manual

428
**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the
Splunk community has using the lookup command.

**makecontinuous**

**Description**

Makes a field on the x-axis numerically continuous by adding empty buckets for
periods where there is no data and quantifying the periods where there is data.
This x-axis field can then be invoked by the `chart` and `timechart` commands.

**Syntax**

`makecontinuous [<field>] <bins-options>...`

**Required arguments**

`<bins-options>`

- **Datatype:** bins | span | start-end
- **Description:** Discretization options. See "Bins options" for details.

**Optional arguments**

`<field>`

- **Datatype:** <field>
- **Description:** Specify a field name.

**Bins options**

`bins`

- **Syntax:** `bins=<int>`
- **Description:** Sets the maximum number of bins to discretize into.

`span`

- **Syntax:** `<log-span> | <span-length>`
- **Description:** Sets the size of each bin, using a span length based on time
  or log-based span.

`<start-end>`
Syntax: end=<num> | start=<num>
Description: Sets the minimum and maximum extents for numerical bins. Data outside of the [start, end] range is discarded.

**Span options**

<log-span>
Syntax: [<num>]log[<num>]
Description: Sets to log-based span. The first number is a coefficient. The second number is the base. If the first number is supplied, it must be a real number >= 1.0 and < base. Base, if supplied, must be real number > 1.0, meaning it must be strictly greater than 1.

span-length
Syntax: <span>[<timescale>]
Description: A span length based on time.

<span>
Syntax: <int>
Description: The span of each bin. If using a timescale, this is used as a time range. If not, this is an absolute bin "length."

<timescale>
Syntax: <sec> | <min> | <hr> | <day> | <month> | <subseconds>
Description: Time scale units.

<table>
<thead>
<tr>
<th>Time scale</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sec&gt;</td>
<td>s</td>
<td>sec</td>
</tr>
<tr>
<td>&lt;min&gt;</td>
<td>m</td>
<td>min</td>
</tr>
<tr>
<td>&lt;hr&gt;</td>
<td>h</td>
<td>hr</td>
</tr>
<tr>
<td>&lt;day&gt;</td>
<td>d</td>
<td>day</td>
</tr>
<tr>
<td>&lt;month&gt;</td>
<td>mon</td>
<td>month</td>
</tr>
<tr>
<td>&lt;subseconds&gt;</td>
<td>us</td>
<td>ms</td>
</tr>
</tbody>
</table>
### Usage

The `makecontinuous` command is a **transforming command**. See [Command types](#).

### Examples

**Example 1:**

Make "_time" continuous with a span of 10 minutes.

```plaintext
... | makecontinuous _time span=10m
```

### See also

chart, timechart

### Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `makecontinuous` command.

### makemv

#### Description

Converts a single valued field into a multivalue field by splitting the values on a simple string delimiter. The delimiter can be a multicharacter delimiter. Alternatively, splits field by using a `regex`.

#### Syntax

```plaintext
makemv [delim=<string> | tokenizer=<string>] [allowempty=<bool>] [setsv=<bool>] <field>
```
**Required arguments**

field

**Syntax:** `<field>`

**Description:** Specify the name of a field.

**Optional arguments**

delim

**Syntax:** `delim=<string>`

**Description:** A string value used as a delimiter. Splits the values in `field` on every occurrence of this string.

**Default:** A single space (" ").

tokenizer

**Syntax:** `tokenizer=<string>`

**Description:** A regex, with a capturing group, that is repeat-matched against the text of `field`. For each match, the first capturing group is used as a value of the newly created multivalue field.

allowempty

**Syntax:** `allowempty=<bool>`

**Description:** Specifies whether to permit empty string values in the multivalue field. When using `delim=true`, repeats of the delimiter string produce empty string values in the multivalue field. For example if `delim=","` and `field="a,,b"`, by default does not produce any value for the empty string. When using the `tokenizer` argument, zero length matches produce empty string values. By default they produce no values.

**Default:** false

setsv

**Syntax:** `setsv=<bool>`

**Description:** If true, the `makemv` command combines the decided values of the field into a single value, which is set on the same field. (The simultaneous existence of a multivalue and a single value for the same field is a problematic aspect of this flag.)

**Default:** false

**Usage**

The `makemv` command is a distributable streaming command. See Command types.
You can use evaluation functions and statistical functions on multivalue fields or to return multivalue fields.

Examples

1. Use a comma to separate field values

For sendmail search results, separate the values of "senders" into multiple values. Display the top values.

    `eventtype="sendmail" | makemv delim="," senders | top senders`

2. Use a colon delimiter and allow empty values

Separate the value of "foo" into multiple values.

    `... | makemv delim=":" allowempty=true foo`

3. Use a regular expression to separate values

The following search creates a result and adds three values to the `my_multival` field. The `makemv` command is used to arrange the values into separate lines.

    `| makeresults | eval my_multival="one,two,three" | makemv tokenizer="(^[^,]+),?,?" my_multival`

See also

**Commands:**
- `mvcombine`
- `mvexpand`
- `nomv`

**Functions:**
- Multivalue eval functions
- Multivalue stats and chart functions
- `split`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the makemv command.
makeresults

Description

Generates the specified number of search results.

If you do not specify any of the optional arguments, this command runs on the local machine and generates one result with only the _time field.

Syntax

| makeresults [<count>] [<annotate>] [<splunk-server>] [<splunk-server-group>...]

Required arguments

None.

Optional arguments

<count>

Syntax: count=<num>
Description: The number of results to generate. If you do not specify the annotate argument, the results have only the _time field.
Default: 1

<annotate>

Syntax: annotate=<bool>
Description: If annotate=true, generates results with the fields shown in the table below.
If annotate=false, generates results with only the _time field.
Default: false

Fields generated with annotate=true

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_raw</td>
<td>None.</td>
</tr>
<tr>
<td>_time</td>
<td>Date and time that you run the makeresults command.</td>
</tr>
<tr>
<td>host</td>
<td>None.</td>
</tr>
</tbody>
</table>
You can use these fields to compute aggregate statistics.

<splunk-server>
  Syntax: splunk_server=<string>
  Description: Use to generate results on one specific server. Use 'local' to refer to the search head.
  Default: local. See the Usage section.

<splunk-server-group>
  Syntax: (splunk_server_group=<string>)...
  Description: Use to generate results on a specific server group or groups.
  You can specify more than one <splunk_server_group>.
  Default: none. See the Usage section.

Usage

The makeresults command is a report-generating command. See Command types.

Generating commands use a leading pipe character and should be the first command in a search.

You can use this command with the eval command to generate an empty result for the eval command to operate on. See the Examples section.

Order-sensitive processors might fail if the internal _time field is absent.

Specifying server and server groups

If you use Splunk Cloud, omit any server or server group argument.

If you are using Splunk Enterprise, by default results are generated only on the originating search head, which is equivalent to specifying splunk_server=local.
If you provide a specific `splunk_server` or `splunk_server_group`, then the number of results you specify with the `count` argument are generated on the all servers or server groups that you specify.

If you specify a server, the results are generated for that server, regardless of the server group that the server is associated with.

If you specify a count of 5 and you target 3 servers, then you will generate 15 total results. If `annotate=true`, the names for each server appear in the `splunk_server` column. This column will show that each server produced 5 results.

**Examples**

1. **Create a result as an input into the `eval` command**

   Sometimes you want to use the `eval` command as the first command in a search. However, the `eval` command expects events as inputs. You can create a dummy event at the beginning of a search by using the `makeresults` command. You can then use the `eval` command in your search.

   ```
   | makeresults | eval newfield="avalue"
   ```

2. **Determine if the modified time of an event is greater than the relative time**

   For events with the field `scheduled_time` that is in Unix Epoch time, determine if the scheduled time is greater than the relative time. The relative time is 1 minute before now. This search uses a subsearch that starts with the `makeresults` command.

   ```
   index=_internal sourcetype=scheduler ( scheduled_time > [ makeresults | eval it=relative_time(now(), "-m") | return $it ] )
   ```

**See also**

`gentimes`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the `makeresults` command.
map

Description

The map command is a looping operator that runs a search repeatedly for each input event or result. You can run the map command on a saved search or an ad hoc search.

Syntax

map (<searchoption> | <savedsplunkoption>) [maxsearches=int]

Required arguments

<savedsplunkoption>
   Syntax: <string>
   Description: Name of a saved search to run for each input result.
   Default: No default.

<searchoption>
   Syntax: search="<string>"
   Description: An ad hoc search to run for each input result. For example:
   ... | map search="search index=_internal earliest=$myearliest$ latest=$mylatest$".
   Default: No default.

Optional arguments

maxsearches
   Syntax: maxsearches=<int>
   Description: The maximum number of searches to run. A message is generated if there are more search results than the maximum number that you specify.
   Default: 10

Usage

Known limitations

You cannot use the map command after an append or appendpipe command in your search pipeline.
**Variable for field names**

When using a saved search or a literal search, the `map` command supports the substitution of `$variable$` strings that match field names in the input results. A search with a string like `$count$`, for example, will replace the variable with the value of the `count` field in the input search result.

When using the `map` command in a dashboard `<form>`, use double dollar signs (`$$`) to specify a variable string. For example, `$$count$$`. See Dashboards and forms.

**Search ID field**

The `map` command also supports a search ID field, provided as `_serial_id$`. The search ID field will have a number that increases incrementally each time that the search is run. In other words, the first run search will have the ID value 1, and the second 2, and so on.

**Basic examples**

1. **Invoke the map command with a saved search**

   ```
   error | localize | map mytimebased_savedsearch
   ```

2. **Map the start and end time values**

   ```
   ... | map search="search starttimeu::$start$ endtimeu::$end$" maxsearches=10
   ```

**Extended examples**

1. **Use a Sudo event to locate the user logins**

   This example illustrates how to find a Sudo event and then use the `map` command to trace back to the computer and the time that users logged on before the Sudo event. Start with the following search for the Sudo event.

   ```
   sourcetype=syslog sudo | stats count by user host
   ```

   This search returns a table of results.

<table>
<thead>
<tr>
<th>User</th>
<th>Host</th>
<th>Count</th>
</tr>
</thead>
</table>
Pipe these results into the `map` command, substituting the username.

```
sourcetype=syslog sudo | stats count by user host | map search="search index=ad_summary username=$user$ type_logon=ad_last_logon"
```

It takes each of the three results from the previous search and searches in the `ad_summary` index for the logon event for the user. The results are returned as a table.

<table>
<thead>
<tr>
<th>_time</th>
<th>computername</th>
<th>computertime</th>
<th>username</th>
<th>usertime</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/12/2016</td>
<td>Workstation$</td>
<td>10/12/2016 08:25:42</td>
<td>userA</td>
<td>10/12/2016 08:31:35 AM</td>
</tr>
</tbody>
</table>

(Thanks to Splunk user Alacercogitatus for this example.)

**See also**

gentimes, search

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `map` command.

**mcollect**

**Description**

Converts events into metric data points and inserts the metric data points into a metric index on the search head.

If you are forwarding data to the indexer, your data will be inserted on the indexer instead of the search head.
Syntax

mcollect index=<string> [file=<string>] [split=<bool>] [spool=<bool>] [prefix_field=<string>] [host=<string>] [source=<string>] [sourcetype=<string>] [<field-list>]

**Required arguments**

`index`

**Syntax:** index=<string>

**Description:** Name of the metric index where the collected metric data is added.

`field-list`

**Syntax:** <field>, ...

**Description:** A list of dimension fields. Required if split=true. Optional if split=false. If unspecified, which implies that split=false, all fields are treated as dimensions for the data point except for the metric_name, prefix_field, and all internal fields.

**Default:** No default value

**Optional arguments**

`file`

**Syntax:** file=<string>

**Description:** The file name where you want the collected metric data to be written. Only applicable when spool=false. You can use a timestamp or a random number for the file name by specifying either file=$timestamp$ or file=$random$.

**Default:** $random$_metrics.csv

`split`

**Syntax:** split=<bool>

**Description:** If set to false, the results must include a metric_name field for the name of the metric and a _value field for the numerical value of the metric. If set to true, then <field-list> must be specified.

**Default:** false

`spool`

**Syntax:** spool=<bool>

**Description:** If set to true, the metrics data file is written to the Splunk spool directory, $SPLUNK_HOME/var/spool/splunk, where the file is indexed. Once the file is indexed, it is removed. If set to false, the file is
written to the $$SPLUNK_HOME/var/run/splunk$$ directory. The file remains in this directory unless further automation or administration is done.

**Default**: true

**prefix_field**

**Syntax**: prefix_field=<string>

**Description**: Only applicable when split=true. If specified, any data point with that field missing is ignored. Otherwise, the field value is prefixed to the metric name.

**Default**: No default value

**host**

**Syntax**: host=<string>

**Description**: The name of the host that you want to specify for the collected metrics data. Only applicable when spool=true.

**Default**: No default value

**source**

**Syntax**: source=<string>

**Description**: The name of the source that you want to specify for the collected metrics data.

**Default**: If the search is scheduled, the name of the search. If the search is ad-hoc, the name of the file that is written to the var/spool/splunk directory containing the search results.

**sourcetype**

**Syntax**: sourcetype=<string>

**Description**: The name of the source type that is specified for the collected metrics data. The Splunk platform does not calculate license usage for data indexed with mcollect_stash, the default source type. If you change the value of this setting to a different source type, the Splunk platform calculates license usage for any data indexed by the mcollect command.

**Default**: mcollect_stash

Do not change this setting without assistance from Splunk Professional Services or Splunk Support. Changing the source type requires a change to the props.conf file.

**Usage**

You use the mcollect command to convert events into metric data to be stored in a metric index on the search head. The metrics data uses a specific format for the metrics fields. See Metrics data format in *Metrics*. 
The `mcollect` command causes new data to be written to a metric index for every run of the search.

**Splitting**

If each result contains only one `metric_name` field and one numeric `_value` field, then the result is a normalized metric data point. This result can be consumed directly and does not need to be split. Otherwise, each result is split into multiple metric data points based on the specified list of dimension fields.

For example, if you have the following data:

```
type=cpu usage=0.78 idle=0.22
```

You have two metrics, `usage` and `idle`.

If you specify in your search:

```
split=true prefix_field=type
```

The results are that the value of the field you specify is used as a prefix to the metric field names. In this case, because `type` is specified, its value, `cpu`, becomes the metric name prefix.

```
metric_name_value
cpu.usage 0.78
cpu.idle 0.22
```

**Time**

If the `_time` field is present in the results, it is used as the timestamp of the metric data point. If the `_time` field is not present, the current time is used.

**field-list**

If `field-list` is not specified, all fields are treated as dimensions for the data point, except for the `prefix_field` and internal fields (fields with an underscore `_` prefix). If `field-list` is specified, the list must appear at the end of the `mcollect` command arguments. If `field-list` is specified, all fields are treated as metric values, except for the fields in `field-list`, the `prefix-field`, and internal fields.

The name of each metric value is the field name prefixed with the `prefix_field` value.
Effectively, one metric data point is returned for each qualifying field that contains a numerical value. If one search result contains multiple qualifying metric name/value pairs, the result is split into multiple metric data points.

**Examples**

**1: Generate a count of error events as metric data points**

The following example shows you how to generate a count of error events and convert them into metric data points in a metric index called 'my_metric_index'.

```
ERROR | stats count BY type | rename count AS _value type AS metric_name 
| mcollect index=my_metric_index
```

**2: Split data into multiple metrics with a common dimension**

The following example shows you how to generate metric data by splitting your data into two metrics (avg.bytes and avg.response_time) with a common dimension (uri).

```
sourcetype=access_combined | stats avg(bytes) AS avg.bytes avg(response_time) AS avg.response_time BY uri | mcollect index=my_metric_index split=t uri
```

**See also**

**Commands**
- collect
- meventcollect

**metadata**

**Description**

The `metadata` command returns a list of sources, sourcetypes, or hosts from a specified index or distributed search peer. The `metadata` command returns information accumulated over time. You can view a snapshot of an index over a specific timeframe, such as the last 7 days, by using the time range picker.

See **Usage**.
Syntax

| metadata type=<metadata-type> [index-specifier]...
[splunk_server=<wc-string>] [splunk_server_group=<wc-string>]

**Required arguments**

type

**Syntax:** type= hosts | sources | sourcetypes

**Description:** The type of metadata to return. This must be one of the three literal strings: hosts, sources, or sourcetypes.

**Optional arguments**

index-specifier

**Syntax:** index=<index_name>

**Description:** Specifies the index from which to return results. You can specify more than one index. Wildcard characters (*) can be used. To match non-internal indexes, use `index=*`. To match internal indexes, use `index=_*`.

**Example:**

```
| metadata type=hosts index=cs* index=na* index=ap* index=eu*
```

**Default:** The default index, which is usually the **main** index.

splunk_server

**Syntax:** splunk_server=<wc-string>

**Description:** Specifies the distributed search peer from which to return results. If you are using Splunk Cloud, omit this parameter. If you are using Splunk Enterprise, you can specify only one `splunk_server` argument. However, you can use a wildcard when you specify the server name to indicate multiple servers. For example, you can specify `splunk_server=peer01` or `splunk_server=peer*`. Use `local` to refer to the search head.

**Default:** All configured search peers return information

splunk_server_group

**Syntax:** splunk_server_group=<wc-string>

**Description:** Limits the results to one or more server groups. If you are using Splunk Cloud, omit this parameter. You can specify a wildcard character in the string to indicate multiple server groups.
Usage

The `metadata` command is a report-generating command. See Command types.

Generating commands use a leading pipe character and should be the first command in a search.

Although the `metadata` command fetches data from all peers, any command run after it runs only on the search head.

The command shows the first, last, and most recent events that were seen for each value of the specified `metadata` type. For example, if you search for:

```
| metadata type=hosts
```

Your results should look something like this:

<table>
<thead>
<tr>
<th></th>
<th>1stTime</th>
<th>Host</th>
<th>2ndTime</th>
<th>recentTime</th>
<th>totalcount</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12/09/19 08:56</td>
<td>apache1</td>
<td>12/09/19 08:56</td>
<td>12/09/19 08:56</td>
<td>27936</td>
<td>hosts</td>
</tr>
<tr>
<td>2</td>
<td>12/09/19 08:56</td>
<td>apache2</td>
<td>12/09/19 08:56</td>
<td>12/09/19 08:56</td>
<td>27936</td>
<td>hosts</td>
</tr>
<tr>
<td>3</td>
<td>12/09/19 08:56</td>
<td>apache3</td>
<td>12/09/19 08:56</td>
<td>12/09/19 08:56</td>
<td>9189</td>
<td>hosts</td>
</tr>
<tr>
<td>4</td>
<td>12/09/19 08:56</td>
<td>apache4</td>
<td>12/09/19 08:56</td>
<td>12/09/19 08:56</td>
<td>9189</td>
<td>hosts</td>
</tr>
</tbody>
</table>

- The `firstTime` field is the timestamp for the first time that the indexer saw an event from this host.
- The `lastTime` field is the timestamp for the last time that the indexer saw an event from this host.
- The `recentTime` field is the indextime for the most recent time that the index saw an event from this host. In other words, this is the time of the last update.
- The `totalcount` field is the total number of events seen from this host.
- The `type` field is the specified type of metadata to display. Because this search specifies `type=hosts`, there is also a `host` column.

In most cases, when the data is streaming live, the `lastTime` and `recentTime` field values are equal. If the data is historical, however, the values might be different.

In small testing environments, the data is complete. However, in environments with large numbers of values for each category, the data might not be complete. This is intentional and allows the `metadata` command to operate within reasonable time and memory usage.
**Time ranges**

Set the time range using the Time Range Picker. You cannot use the earliest or latest time range modifiers in the search string. Time range modifiers must be set before the first piped command and generating commands in general do not allow anything to be specified before the first pipe.

If you specify a time range other than All Time for your search, the search results might not be precise. The metadata is stored as aggregate numbers for each bucket on the index. A bucket is either included or not included based on the time range you specify.

For example, you run the following search specifying a time range of **Last 7 days.** The time range corresponds to January 1st to January 7th.

```
| metadata type=sourcetypes index=ap
```

There is a bucket on the index that contains events from both December 31st and January 1st. The metadata from that bucket is included in the information returned from search.

**Maximum results**

By default, a maximum of 10,000 results are returned. This maximum is controlled by the `maxresultrows` setting in the `[metadata]` stanza in the `limits.conf` file.

**Examples**

1. **Search multiple indexes**

Return the metadata for indexes that represent different regions.

```
| metadata type=hosts index=cs* index=na* index=ap* index=eu*
```

2. **Search for sourcetypes**

Return the values of sourcetypes for events in the `_internal` index.

```
| metadata type=sourcetypes index=_internal
```

This returns the following report.

446
3. Format the results from the metadata command

You can also use the `fieldformat` command to format the results of the `firstTime`, `lastTime`, and `recentTime` columns to be more readable.

```bash
| metadata type=sourcetypes index=_internal | rename totalCount as Count firstTime as "First Event" lastTime as "Last Event" recentTime as "Last Update" | fieldformat Count=tostring(Count, "commas") | fieldformat "First Event"=strftime('First Event', "%c") | fieldformat "Last Event"=strftime('Last Event', "%c") | fieldformat "Last Update"=strftime('Last Update', "%c")
```

Click on the `Count` field label to sort the results and show the highest count first.

Now, the results are more readable:

4. Return values of "sourcetype" for events in a specific index on a specific server

Return values of "sourcetype" for events in the "_audit" index on server foo.

```bash
| metadata type=sourcetypes index=_audit splunk_server=foo
```

See also

`dbinspect`

`tstats`
Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the metadata command.

metasearch

Description

Retrieves event metadata from indexes based on terms in the <logical-expression>. Metadata fields include source, sourcetype, host, _time, index, and splunk_server.

Syntax

metasearch [<logical-expression>]

Optional arguments

<logical-expression>
   Syntax: <time-opts>|<search-modifier>|(NOT)? <logical-expression>|<index-expression>|<comparison-expression>|((<logical-expression>)(OR)? <logical-expression>)
   Description: Includes time and search modifiers, comparison and index expressions.

Logical expression

<comparison-expression>
   Syntax: <field><cmp><value>
   Description: Compare a field to a literal value or values of another field.

<index-expression>
   Syntax: "<string>"|<term>|<search-modifier>

<time-opts>
   Syntax: (<timeformat>)? (<time-modifier>)*
**Comparison expression**

<cmp>

**Syntax:** = | != | < | <= | > | >=
**Description:** Comparison operators.

<field>

**Syntax:** <string>
**Description:** The name of a field. In metasearch, only the fields source, sourcetype, host, _time, index, and splunk_server can be used.

<lit-value>

**Syntax:** <string> | <num>
**Description:** An exact, or literal, value of a field that is used in a comparison expression.

<value>

**Syntax:** <lit-value> | <field>
**Description:** In comparison-expressions, the literal value of a field or another field name where "literal" means number or string.

**Index expression**

<search-modifier>

**Syntax:** <field-specifier>|<savedsplunk-specifier>|<tag-specifier>

**Time options**

The search allows many flexible options for searching based on time. For a list of time modifiers, see the topic "Time modifiers for search" in the *Search Manual*.

<timeformat>

**Syntax:** timeformat=<string>
**Description:** Set the time format for starttime and endtime terms. By default, timestamp is formatted: timeformat=%m/%d/%Y:%H:%M:%S.

<time-modifier>

**Syntax:** earliest=<time_modifier> | latest=<time_modifier>
**Description:** Specify start and end times using relative or absolute time. For more about the time modifier index, see "Specify time modifiers in your search" in the *Search Manual*. 

449
Examples

Example 1:

Return metadata for events with "404" and from host "webserver1".

... | metasearch 404 host=webserver1

See also

metadata, search

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the metasearch command.

meventcollect

Description

Converts events into metric data points and inserts the data into a metric index on the indexers.

Syntax

meventcollect index=<string> [split=<bool>] [spool=<bool>] [prefix_field=<string>] [host=<string>] [source=<string>] [sourcetype=<string>] [<field-list>]

Required arguments

index

Syntax: index=<string>
Description: Name of the metric index where the collected metric data is added.

field-list

Syntax: <field>, ...
Description: A list of dimension fields. Required if split=true. Optional if split=false. If unspecified (which implies that split=false), all fields are treated as dimensions for the data point, except for the metric_name.
prefix_field, and all internal fields.
Default: No default value

Optional arguments

split
Syntax: split=<bool>
Description: If set to false, the results must include a metric_name field for the name of the metric and a _value field for the numerical value of the metric. If set to true, then <field-list> must be specified.
Default: false

spool
Syntax: spool=<bool>
Description: If set to true, the metrics data file is written to the Splunk spool directory, $SPLUNK_HOME/var/spool/splunk, where the file is indexed automatically. If set to false, the file is written to the $SPLUNK_HOME/var/run/splunk directory. The file remains in this directory unless further automation or administration is done.
Default: true

prefix_field
Syntax: prefix_field=<string>
Description: Only applicable when split=true. If specified, any data point with that field missing is ignored. Otherwise, the field value is prefixed to the metric name.
Default: No default value

host
Syntax: host=<string>
Description: The name of the host that you want to specify for the collected metrics data. Only applicable when spool=true.
Default: No default value

source
Syntax: source=<string>
Description: The name of the source that you want to specify for the collected metrics data.
Default: If the search is scheduled, the name of the search. If the search is ad-hoc, the name of the file that is written to the var/spool/splunk directory containing the search results.
Syntax: `sourcetype=<string>`

Description: The name of the source type that you want to specify for the collected metrics data.

Default: metrics_csv

Do not change this setting without assistance from Splunk Professional Services or Splunk Support. Changing the source type requires a change to the `props.conf` file.

Usage

You use the `meventcollect` command to convert events into metric data to be stored in a metric index on the indexers. The metrics data uses a specific format for the metrics fields. See Metrics data format in `Metrics`.

Only streaming commands can precede the `meventcollect` command so that results can be ingested on the indexers. If you would like to run a search that uses transforming commands to generate metric data points, use `mcollect` instead of `meventcollect`.

The `meventcollect` command causes new data to be written to a metric index for every run of the search.

Splitting

If each result contains only one `metric_name` field and one numeric `_value` field, then the result is a normalized metric data point. This result can be consumed directly and does not need to be split. Otherwise, each result is split into multiple metric data points based on the specified list of dimension fields.

field-list

If `field-list` is not specified, all fields are treated as dimensions for the data point, except for the `prefix_field` and internal fields (fields with an underscore '_' prefix). If `field-list` is specified, the list must appear at the end of the `meventcollect` command arguments. If `field-list` is specified, all fields are treated as metric values, except for the fields in `field-list`, the `prefix-field`, and internal fields.

The name of each metric value is the field name prefixed with the `prefix_field` value.

Effectively, one metric data point is returned for each qualifying field that contains a numerical value. If one search result contains multiple qualifying metric
name/value pairs, the result is split into multiple metric data points.

**Examples**

1: **Collect metrics.log data into a metrics index**

The following example shows you how to collect metrics log data into a metric index called 'my_metric_index'.

```bash
index=_internal source= */metrics.log | eval prefix = group + "." + name | meventcollect index=my_metric_index split=true prefix_field=prefix
```

**See also**

**Commands**

- collect
- mcollect

**mstats**

**Description**

Use the `mstats` command to analyze metrics. This command performs statistics on the measurement, metric_name, and dimension fields in metric indexes. You can use `mstats` in **historical searches** and **real-time searches**. When you use `mstats` in a real-time search with a time window, a historical search runs first to backfill the data.

The `mstats` command provides the best search performance when you use it to search a single metric_name value or a small number of metric_name values.

**Syntax**

```bash
| mstats [prestats=<bool>] [append=<bool>] [backfill=<bool>] [update_period=<integer>] <stats-metric-term>... WHERE [<logical-expression>]... [ (BY|GROUPBY) <field-list> ] [ <span-length> ]
```
Required arguments

<stats-metric-term>

Syntax: (<stats-func>"("<metric_name>")" [AS <string>])...

Description: Perform statistical calculations on one or more metric_name fields. You can rename the result of each function using the AS clause, unless prestats is set to true. The metric_name must be in parenthesis.

The following table lists the supported functions for the mstats command by type of function. Use the links in the table to see descriptions and examples for each function.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate functions</td>
<td>avg()</td>
</tr>
<tr>
<td></td>
<td>count()</td>
</tr>
<tr>
<td></td>
<td>max()</td>
</tr>
<tr>
<td></td>
<td>median()</td>
</tr>
<tr>
<td></td>
<td>min()</td>
</tr>
<tr>
<td>Time functions</td>
<td>earliest()</td>
</tr>
<tr>
<td></td>
<td>earliest_time()</td>
</tr>
</tbody>
</table>

For an overview of using functions with commands, see Statistical and charting functions.

Optional arguments

append

Syntax: append=<bool>

Description: Valid only when prestats=true. This argument runs the mstats command and adds the results to an existing set of results instead of generating new results.

Default: false

backfill

Syntax: backfill=<bool>

Description: Valid only with real-time searches that have a time window. When backfill=true, the mstats command runs a search on historical data to backfill events before searching the in-memory real-time data.

Default: true
<field-list>
  Syntax: <field>, ...
  Description: Specifies one or more fields to group the results by.
              Required when using the BY or GROUPBY clause.
</field-list>

<logical-expression>
  Syntax: <time-opts>|<search-modifier>|(NOT)?
          <logical-expression>||<index-expression>|<comparison-expression>|<(logical-expression>
          (OR)? <logical-expression>)
  Description: An expression describing the filters that are applied to your
              search. Includes time and search modifiers, comparison expressions, and
              index expressions. See the following sections for descriptions of each of
              these logical expression components.
              Cannot filter on metric_name. Does not support CASE or TERM directives.
              You also cannot use the WHERE clause to search for terms or phrases.
</logical-expression>

prestats
  Syntax: prestats=true | false
  Description: Specifies whether to use the prestats format. The prestats
              format is a Splunk internal format that is designed to be consumed by
              commands that generate aggregate calculations. When you use the
              prestats format, you can pipe the data into the chart, stats, or timechart
              commands, which are designed to accept the prestats format. When
              prestats is set to true, instructions with the AS clause are not relevant.
              The field names for the aggregates are determined by the command that
              consumes the prestats format and produces the aggregate output.
  Default: false

<span-length>
  Syntax: span=<int><timescale>
  Description: The span of each time bin. If used with a <timescale>, the
              span-length is treated as a time range. If not, this is an absolute bucket
              length. If you do not specify a <span-length>, the default is auto, which
              means that the number of time buckets adjusts to produce a reasonable
              number of results. For example, if seconds are used initially for the
              <timespan> and too many results are returned, the <timescale> is
              changed to a longer value, such as minutes, to return fewer time buckets.
</span-length>

<timescale>
  Syntax: <sec> | <min> | <hr> | <day> | <month>
  Description: Time scale units. For the mstats command, the
              <timescale> does not support subseconds.
  Default: sec
**Time scale Syntax Description**

<table>
<thead>
<tr>
<th>Time scale</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sec&gt;</td>
<td>s</td>
<td>sec</td>
</tr>
<tr>
<td></td>
<td>Time scale in seconds.</td>
<td></td>
</tr>
<tr>
<td>&lt;min&gt;</td>
<td>m</td>
<td>min</td>
</tr>
<tr>
<td></td>
<td>Time scale in minutes.</td>
<td></td>
</tr>
<tr>
<td>&lt;hr&gt;</td>
<td>h</td>
<td>hr</td>
</tr>
<tr>
<td></td>
<td>Time scale in hours.</td>
<td></td>
</tr>
<tr>
<td>&lt;day&gt;</td>
<td>d</td>
<td>day</td>
</tr>
<tr>
<td></td>
<td>Time scale in days.</td>
<td></td>
</tr>
<tr>
<td>&lt;month&gt;</td>
<td>mon</td>
<td>month</td>
</tr>
<tr>
<td></td>
<td>Time scale in months.</td>
<td></td>
</tr>
</tbody>
</table>

**update_period**

*Syntax:* update_period=<integer>

*Description:* Valid only with real-time searches. Specifies how frequently, in milliseconds, the real-time summary for the `mstats` command is updated. A larger number means less frequent updates to the summary and less impact on index processing.

*Default:* 1000 (1 second)

**Logical expression options**

<comparison-expression>

*Syntax:* `<field>`<comparison-operator><value> | `<field>` IN (<value-list>)

*Description:* Compares a field to a literal value or provides a list of values that can appear in the field.

<index-expression>

*Syntax:* `<term>` | `<search-modifier>`

*Description:* Describes the events you want to retrieve from the index using search terms and search modifiers.

<time-opts>

*Syntax:* `[<timeformat>] (<time-modifier>)*

*Description:* Describes the format of the `<starttime>` and `<endtime>` terms of the search.

**Comparison expression options**

<comparison-operator>

*Syntax:* = | != | < | <= | > | >=

*Description:* You can use comparison operators when searching field-value pairs. Comparison expressions with the equal ( = ) or not equal ( != ) operator compare string values. For example, "1" does not
match "1.0". Comparison expressions with greater than or less than operators < > <= >= numerically compare two numbers and lexicographically compare other values. See Usage.

<field>
  Syntax: <string>
  Description: The name of a field.

<value>
  Syntax: <literal-value>
  Description: In comparison expressions, this is the literal number or string value of a field.

<value-list>
  Syntax: (<literal-value>, <literal-value>, ...)
  Description: Used with the IN operator to specify two or more values. For example use error IN (400, 402, 404, 406) instead of error=400 OR error=402 OR error=404 OR error=406

Index expression options

<string>
  Syntax: "<string>"
  Description: Specify keywords or quoted phrases that the search must match. When searching for strings, quoted strings, or anything that is not a search modifier, Splunk software searches the _raw field for the matching events or results.

<search-modifier>
  Syntax: <sourcetype-specifier> | <host-specifier> | <source-specifier> | <splunk_server-specifier>
  Description: Search for events from specified fields. For example, search for one or a combination of hosts, sources, and source types. See searching with default fields in the Knowledge Manager manual.

<sourcetype-specifier>
  Syntax: sourcetype=<string>
  Description: Search for events from the specified sourcetype field.

<host-specifier>
  Syntax: host=<string>
  Description: Search for events from the specified host field.
<source-specifier>
Syntax: source=<string>
Description: Search for events from the specified source field.

<splunk_server-specifier>
Syntax: splunk_server=<string>
Description: Search for events from a specific server. Use "local" to refer to the search head.

Time options
<timeformat>
Syntax: timeformat=<string>
Description: Set the time format for starttime and endtime terms.
Default: timeformat=%m/%d/%Y:%H:%M:%S.

<time-modifier>
Syntax: starttime=<string> | endtime=<string> | earliest=<time_modifier> | latest=<time_modifier>
Description: Specify start and end times using relative or absolute time.

You can also use the earliest and latest attributes to specify absolute and relative time ranges for your search. For more about this time modifier syntax, see About search time ranges in the "Search Manual".

starttime
Syntax: starttime=<string>
Description: Events must be later or equal to this time. The starttime match the <code>timeformat</code>.

datetime
Syntax: endtime=<string>
Description: All events must be earlier or equal to this time.

For a list of time modifiers, see Time modifiers for search.

Usage

The mstats command is a report-generating command, except when append=true. See Command types.

Generating commands use a leading pipe character and should be the first command in a search, except when append=true is specified with the command.
Use the `mstats` command to search metrics data. The metrics data uses a specific format for the metrics fields. See Metrics data format in *Metrics*.

**Wildcard characters**

The `mstats` command supports wildcard characters in any search filter. However, you cannot use wildcard characters in the GROUP BY clause.

If you are using wildcard characters in your aggregations and you are renaming them, your rename must have matching wildcards.

For example, this search is invalid:

```
| mstats sum(*.free) as FreeSum
```

This search is valid:

```
| mstats sum(*.free) as *FreeSum
```

**Aggregations**

Numeric aggregations are only allowed on specific values of the `metric_name` field. The metric name must be enclosed in parenthesis. If there is no data for the specified `metric_name` in parenthesis, the search is still valid.

Aggregations are not allowed for values of any other field, including the `_time` field.

**WHERE clause**

Use the WHERE clause to filter by any of the supported dimensions.

The WHERE clause cannot filter by `metric_name`. Filtering by `metric_name` is performed based on the `metric_name` fields specified with the `<stats-metric-term>` argument.

If you do not specify an index name in the WHERE clause, the `mstats` command returns results from the default metrics indexes associated with your role. If you do not specify an index name and you have no default metrics indexes associated with your role, `mstats` returns no results. To search against all metrics indexes use `WHERE index=*`. 

459
For more information about defining default metrics indexes for a role, see Add and edit roles with Splunk Web in *Securing Splunk Enterprise*.

**Grouping results**

You can group results by the dimension and metric_name fields.

You can also group by time. You must specify a timespan using the <span-length> argument to group by time buckets. For example, span=1hr or span=auto. The <span-length> argument is separate from the BY clause and can be placed at any point in the search between clauses.

Grouping by the _value or _time fields is not allowed.

**Time dimensions**

The mstats command does not recognize the following time-related dimensions.

<table>
<thead>
<tr>
<th>Unsupported dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_hour</td>
</tr>
<tr>
<td>date_mday</td>
</tr>
<tr>
<td>date_minute</td>
</tr>
<tr>
<td>date_month</td>
</tr>
<tr>
<td>date_second</td>
</tr>
<tr>
<td>date_wday</td>
</tr>
<tr>
<td>date_year</td>
</tr>
<tr>
<td>date_zone</td>
</tr>
<tr>
<td>metric_timestamp</td>
</tr>
<tr>
<td>time</td>
</tr>
<tr>
<td>timeendpos</td>
</tr>
<tr>
<td>timestamp</td>
</tr>
<tr>
<td>timestartpos</td>
</tr>
</tbody>
</table>

**Lexicographical order**

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.

- Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically as 10, 100, 70, 9.
- Uppercase letters are sorted before lowercase letters.
- Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

You can specify a custom sort order that overrides the lexicographical order. See the blog *Order Up! Custom Sort Orders*. 

460
**Deprecated syntax**

In version 7.1 of the Splunk platform, elements of the `mstats` syntax were deprecated and replaced with an improved syntax. The deprecated syntax has the following requirements:

- All statistical functions must be performed on the `_value` field.
- A single `metric_name` filter must be specified for the `WHERE` clause.

The replacement syntax makes it much easier to perform statistical aggregations across multiple metrics in a single `mstats` search.

**Examples**

1. **Calculate a single metric grouped by time**

   Return the average value of the `aws.ec2.CPUUtilization` metric in the `mymetricdata` metric index. Bucket the results into 30 second time spans.

   ```bash
   | mstats avg(aws.ec2.CPUUtilization) WHERE index=mymetricdata span=30s
   ```

2. **Combine metrics with different metric names**

   Return the average value of both the `aws.ec2.CPUUtilization` metric and the `os.cpu.utilization` metric. Group the results by host and bucket the results into 1 minute time spans. Both metrics are combined and considered a single metric series.

   ```bash
   | mstats avg(aws.ec2.CPUUtilization) avg(os.cpu.utilization) WHERE index=mymetricdata BY host span=1m
   ```

3. **Use prestats=t mode with the timechart command**

   Return a timechart of the number of `aws.ec2.CPUUtilization` metric data points for each day.

   ```bash
   | mstats prestats=t count(aws.ec2.CPUUtilization) WHERE index=mymetricdata span=1d | timechart span=1d count(aws.ec2.CPUUtilization)
   ```

4. **Filter the results on a dimension value and split by the values of another dimension**
Return the average value of the `aws.ec2.CPUUtilization` metric for all measurements with `host=foo` and split the results by the values of the `app` dimension.

```
| mstats avg(aws.ec2.CPUUtilization) WHERE host=foo BY app
```

### 5. Specify multiple aggregations of multiple metrics

Return the average and maximum of the resident set size and virtual memory size. Group the results by `metric_name` and bucket them into 1 minute spans.

```
| mstats avg(os.mem.rss) AS "AverageRSS" max(os.mem.rss) AS "MaxRSS"
  avg(os.mem.vsz) AS "AverageVMS" max(os.mem.vsz) AS "MaxVMS" WHERE
  index=mymetricdata BY metric_name span=1m
```

### 6. Aggregate a metric across all of your default metrics indexes

Find the median of the `aws.ec2.CPUUtilization` metric. Do not include an index filter to search for measurements in all of the default metrics indexes associated with your role.

```
| mstats median(aws.ec2.CPUUtilization)
```

### 7. Deprecated syntax example

Use the deprecated syntax to get a count of all of the measurements for the `aws.ec2.CPUUtilization` metric in the `mymetricdata` index.

```
| mstats count(_value) WHERE metric_name=aws.ec2.CPUUtilization AND
  index=mymetricdata
```

**See also**

Overview of metrics in *Metrics*

**multikv**

**Description**

Extracts field-values from table-formatted events, such as the results of `top`, `netstat`, `ps`, and so on. The `multikv` command creates a new event for each table row and assigns field names from the title row of the table.
An example of the type of data multikv is designed to handle:

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Occupation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Josh</td>
<td>42</td>
<td>SoftwareEngineer</td>
</tr>
<tr>
<td>Francine</td>
<td>35</td>
<td>CEO</td>
</tr>
<tr>
<td>Samantha</td>
<td>22</td>
<td>ProjectManager</td>
</tr>
</tbody>
</table>

The key properties here are:

- Each line of text represents a conceptual record.
- The columns are aligned.
- The first line of text provides the names for the data in the columns.

multikv can transform this table from one event into three events with the relevant fields. It works more easily with the fixed-alignment though can sometimes handle merely ordered fields.

The general strategy is to identify a header, offsets, and field counts, and then determine which components of subsequent lines should be included into those field names. Multiple tables in a single event can be handled (if multitable=true), but may require ensuring that the secondary tables have capitalized or ALLCAPS names in a header row.

Auto-detection of header rows favors rows that are text, and are ALLCAPS or Capitalized.

If you have Splunk Cloud and want to use this feature, file a Support ticket specifying the multi-key-value extractions you want to define.

**Syntax**

multikv [conf=<stanza_name>] [multikv-option>...]

**Optional arguments**

conf

Syntax: conf=<stanza_name>

Description: If you have a field extraction defined in multikv.conf, use this argument to reference the stanza in your search. For more information, refer to the configuration file reference for multikv.conf in the Admin Manual.
Syntax: copyattrs=<bool> | fields <field-list> | filter <term-list> | forceheader=<int> | multitable=<bool> | noheader=<bool> | rmorig=<bool>

Description: Options for extracting fields from tabular events.

Descriptions for multikv options

copyattrs
Syntax: copyattrs=<bool>
Description: When true, multikv copies all fields from the original event to the events generated from that event. When false, no fields are copied from the original event. This means that the events will have no _time field and the UI will not know how to display them.
Default: true

fields
Syntax: fields <field-list>
Description: Limit the fields set by the multikv extraction to this list. Ignores any fields in the table which are not on this list.

filter
Syntax: filter <term-list>
Description: If specified, multikv skips over table rows that do not contain at least one of the strings in the filter list. Quoted expressions are permitted, such as "multiple words" or "trailing_space ".

forceheader
Syntax: forceheader=<int>
Description: Forces the use of the given line number (1 based) as the table's header. Does not include empty lines in the count.
Default: The multikv command attempts to determine the header line automatically.

multitable
Syntax: multitable=<bool>
Description: Controls whether or not there can be multiple tables in a single _raw in the original events.
Default: true

noheader
Syntax: noheader=<bool>
Description: Handle a table without header row identification. The size of the table will be inferred from the first row, and fields will be named Column_1, Column_2, ... noheader=true implies multitable=false.
Default: false

rmorig

Syntax: rmorig=<bool>
Description: When true, the original events will not be included in the output results. When false, the original events are retained in the output results, with each original emitted after the batch of generated results from that original.
Default: true

Usage

The **multikv** command is a distributable streaming command. See [Command types](#).

Examples

**Example 1:** Extract the "COMMAND" field when it occurs in rows that contain "splunkd".

```
... | multikv fields COMMAND filter splunkd
```

**Example 2:** Extract the "pid" and "command" fields.

```
... | multikv fields pid command
```

See also

**extract, kvform, rex, xmlkv**

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the multikv command.

**multisearch**

Description

The **multisearch** command is a generating command that runs multiple *streaming* searches at the same time. This command requires at least two
subsearches and allows only streaming operations in each subsearch. Examples of streaming searches include searches with the following commands: search, eval, where, fields, and rex. For more information, see Types of commands in the Search Manual.

Syntax

| multisearch <subsearch1> <subsearch2> <subsearch3> ...

Required arguments

<subsearch>

Syntax: "["search <logical-expression">"]"
Description: At least two streaming searches must be specified. See the search command for detailed information about the valid arguments for <logical-expression>.

To learn more, see About subsearches in the Search Manual.

Usage

The multisearch command is an event-generating command. See Command types.

Generating commands use a leading pipe character and should be the first command in a search.

Subsearch processing and limitations

With the multisearch command, the events from each subsearch are interleaved. Therefore the multisearch command is not restricted by the subsearch limitations.

Unlike the append command, the multisearch command does not run the subsearch to completion first. The following subsearch example with the append command is not the same as using the multisearch command.

index=a | eval type = "foo" | append [search index=b | eval mytype = "bar"]
Examples

Example 1:

Search for events from both index a and b. Use the `eval` command to add different fields to each set of results.

```
| multisearch [search index=a | eval type = "foo"] [search index=b | eval mytype = "bar"]
```

See also

append, join

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the multisearch command.

mvcombine

Description

Takes a group of events that are identical except for the specified field, which contains a single value, and combines those events into a single event. The specified field becomes a multivalue field that contains all of the single values from the combined events.

There are situations where the `mvjoin` `eval` function is a better option than the `mvcombine` command. See Usage.

Syntax

`mvcombine [delim=<string>] <field>`

Required arguments

field

**Syntax:** `<field>`

**Description:** The name of a field to merge on, generating a multivalue field.
Optional arguments

delim

**Syntax:** delim=<string>

**Description:** Defines the string to use as the delimiter for the values that get combined into the multivalue field. For example, if the values of your field are "1", "2", and "3", and `delim` is ";" then the combined multivalue field is "1";"2";"3".

**Default:** a single space, (" ")

To see the output of the `delim` argument, you must use the `nomv` command immediately after the `mvcombine` command. See Usage

Usage

The `mvcombine` command is a transforming command. See Command types.

You can use evaluation functions and statistical functions on multivalue fields or to return multivalue fields.

The `mvcombine` command accepts a set of input results and finds groups of results where all field values are identical, except the specified field. All of these results are merged into a single result, where the specified field is now a multivalue field.

Because raw events have many fields that vary, this command is most typically useful after paring down the set of available fields with the `fields` command. The command is also useful for manipulating the results of certain reporting commands.

Specifying delimiters

The `mvcombine` command creates a multivalue version of the field you specify, as well as a single value version of the field. The multivalue version is displayed by default.

The single value version of the field is a flat string that is separated by a space or by the delimiter that you specify with the `delim` argument.

By default the multvalue version of the field is displayed in the results. To display the single value version with the delimiters add the `|nomv` command to the end of your search. For example `...| mvcombine delim=""," host | nomv host`. 

468
Some forms modes of investigating the search results prefer this single value representation, such as exporting to CSV in the UI, or running a command line search with `splunk search "..." -output csv`. Some commands that are not multivalue aware might use this single value as well.

Most ways of accessing the search results prefer the multivalue representation, such as viewing the results in the UI, or exporting to JSON, requesting JSON from the command line search with `splunk search "..." -output json` or requesting JSON or XML from the REST API. For these forms of, the selected delim has no effect.

**Using mvjoin instead of mvcombine**

If the field is a multivalue field and you want a single valued field with a different delimiter, use the `mvjoin` evaluation function. For example, a multivalue field contains the values "1","2","3","4","5". You want a single valued field with OR as the delimiter, such as "1 OR 2 OR 3 OR 4 OR 5". Use the `mvjoin` function and not the `mvcombine` command. See Multivalue Eval Functions.

**Examples**

1. **Creating a multivalue field**

   This example uses the sample dataset from the Search Tutorial. To try this example yourself, download the data set from Get the tutorial data into Splunk and follow the instructions in the Search Tutorial to upload the data.

   To understand how mvcombine works, let's explore the data.

   1. Set the time range to **All time**.
   2. Run the following search.

   ```
   index=* | stats max(bytes) AS max, min(bytes) AS min BY host
   ```

   The results show that the **max** and **min** fields have duplicate entries for the hosts that start with **www**. The other hosts show no results for the **max** and **min** fields.
3. To remove the other hosts from your results, modify the search to add `host=www*` to the search criteria.

   ```plaintext
   index=* host=www* | stats max(bytes) AS max, min(bytes) AS min BY host
   ```

   Because the values in the `max` and `min` columns contain the exact same values, you can use the `mvcombine` to combine the host values into a multivalue result.

4. Add `| mvcombine host` to your search and run the search again.

   ```plaintext
   index=* host=www* | stats max(bytes) AS max, min(bytes) AS min BY host | mvcombine host
   ```

   Instead of three rows, one row is returned. The host field is now a multivalue field.

2. **Returning the delimited values**

   As mentioned in the Usage section, by default the delimited version of the results are not returned in the output. To return the results with the delimiters, you must return the single value string version of the field.

   Add the `nomv` command to your search. For example:

   ```plaintext
   index=* host=www* | stats max(bytes) AS max, min(bytes) AS min BY host | mvcombine delim="","" host | nomv host
   ```
The search results that are returned are shown in the following table.

<table>
<thead>
<tr>
<th>host</th>
<th>max</th>
<th>min</th>
</tr>
</thead>
<tbody>
<tr>
<td>www1, www2, www3</td>
<td>4000</td>
<td>200</td>
</tr>
</tbody>
</table>

To return the results with a space after each comma, specify `delim="", "`.  

**Example 3:**

In multivalue events:

```
sourcetype="WMI:WinEventLog:Security" | fields EventCode, Category,RecordNumber | mvcombine delim=""," RecordNumber | nomv RecordNumber
```

**Example 4:**

Combine the values of "foo" with a colon delimiter.

```
... | mvcombine delim=":" foo
```

**See also**

**Commands:**
- `makemv`
- `mvexpand`
- `nomv`

**Functions:**
- Multivalue eval functions
- Multivalue stats and chart functions
- `split`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `mvcombine` command.
**mvexpand**

**Description**
Expands the values of a multivalue field into separate events, one event for each value in the multivalue field. For each result, the `mvexpand` command creates a new result for every multivalue field.

**Syntax**
```
mvexpand <field> [limit=<int>]
```

**Required arguments**
field
- **Syntax:** `<field>`
- **Description:** The name of a multivalue field.

**Optional arguments**
limit
- **Syntax:** `limit=<int>`
- **Description:** Specify the number of values of `<field>` to use for each input event.
- **Default:** 0, or no limit

**Usage**
The `mvexpand` command is a distributable streaming command. See Command types.

You can use evaluation functions and statistical functions on multivalue fields or to return multivalue fields.

**Limits**
A limit exists on the amount of RAM that the `mvexpand` command is permitted to use while expanding a batch of results. By default the limit is 500MB. The input chunk of results is typically `maxresults` or smaller in size, and the expansion of all these results resides in memory at one time. The total necessary memory is the average result size multiplied by the number of results in the chunk multiplied by the average size of the multivalue field being expanded.
If this attempt exceeds the configured maximum on any chunk, the chunk is truncated and a warning message is emitted. If you have Splunk Enterprise, you can adjust the limit by editing the `max_mem_usage_mb` setting in the `limits.conf` file. If you have Splunk Cloud and encounter problems because of this limit, file a Support ticket.

**Examples**

**Example 1:**

Create new events for each value of multivalue field, "foo".

```
... | mvexpand foo
```

**Example 2:**

Create new events for the first 100 values of multivalue field, “foo”.

```
... | mvexpand foo limit=100
```

**Example 3:**

The `mvexpand` command only works on one multivalue field. This example walks through how to expand an event with more than one multivalue field into individual events for each field value. For example, given these events, with sourcetype=data:

```
2018-04-01 00:11:23 a=22 b=21 a=23 b=32 a=51 b=24
2018-04-01 00:11:22 a=1 b=2 a=2 b=3 a=5 b=2
```

First, use the `rex` command to extract the field values for `a` and `b`. Then use the `eval` command and `mvzip` function to create a new field from the values of `a` and `b`.

```
source="mvexpandData.csv" | rex field=_raw "a=(?<a>\d+)" max_match=5 | rex field=_raw "b=(?<b>\d+)" max_match=5 | eval fields = mvzip(a,b) | table _time fields
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-01 00:11:23</td>
<td></td>
</tr>
</tbody>
</table>
Use the `mvexpand` command and the `rex` command on the new field, `fields`, to create new events and extract the alpha and beta values:

```
source="mvexpandData.csv" | rex field=_raw "a=(?<a>\d+)" max_match=5 | rex field=_raw "b=(?<b>\d+)" max_match=5 | eval fields = mvzip(a,b) | mvexpand fields | rex field=fields "(?<alpha>\d+),(?<beta>\d+)" | table
```

<table>
<thead>
<tr>
<th>_time</th>
<th>alpha</th>
<th>beta</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-01 00:11:22</td>
<td>23</td>
<td>32</td>
</tr>
<tr>
<td>2018-04-01 00:11:23</td>
<td>51</td>
<td>24</td>
</tr>
<tr>
<td>2018-04-01 00:11:22</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2018-04-01 00:11:22</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2018-04-01 00:11:22</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

(Thanks to Splunk user Duncan for this example.)

**See also**

**Commands:**
- `makemv`
- `mvcombine`
- `nomv`

**Functions:**
- Multivalue eval functions
- Multivalue stats and chart functions
- `split`
**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the mvexpand command.

**nomv**

**Description**

Converts values of the specified multivalue field into one single value. Overrides the configurations for the multivalue field that are set in the `fields.conf` file.

**Syntax**

`nomv <field>`

**Required arguments**

**field**

**Syntax:** `<field>`

**Description:** The name of a multivalue field.

**Usage**

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

You can use evaluation functions and statistical functions on multivalue fields or to return multivalue fields.

**Examples**

**Example 1:**

For sendmail events, combine the values of the senders field into a single value. Display the top 10 values.

```
eventtype="sendmail" | nomv senders | top senders
```
**See also**

**Commands:**
makemv
mvcombine
mvexpand
convert

**Functions:**
Multivalue eval functions
Multivalue stats and chart functions
split

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the nomv command.

**outlier**

**Description**

This command is used to remove outliers, not detect them. It removes or truncates outlying numeric values in selected fields. If no fields are specified, then the outlier command attempts to process all fields.

Filtering is based on the inter-quartile range (IQR), which is computed from the difference between the 25th percentile and 75th percentile values of the numeric fields. If the value of a field in an event is less than \((25th \text{ percentile}) - \text{param}\cdot\text{IQR}\) or greater than \((75th \text{ percentile}) + \text{param}\cdot\text{IQR}\), that field is transformed or that event is removed based on the action parameter.

To identify outliers and create alerts for outliers, see finding and removing outliers in the *Search Manual*.

**Syntax**

outlier <outlier-options>... [<field-list>]
Optional arguments

<outlier-options>
   Syntax: <action> | <mark> | <param> | <uselower>
   Description: Outlier options.

<field-list>
   Syntax: <field> ...
   Description: A space-delimited list of field names.

Outlier options

&action>
   Syntax: action=remove | transform
   Description: Specifies what to do with the outliers. The remove option removes events that containing the outlying numerical values. The transform option truncates the outlying values to the threshold for outliers. If action=transform and mark=true, prefixes the values with "000".
   Abbreviations: The remove action can be shorted to rm. The transform action can be shorted to tf.
   Default: transform

<mark>
   Syntax: mark=<bool>
   Description: If action=transform and mark=true, prefixes the outlying values with "000". If action=remove, the mark argument has no effect.
   Default: false

<param>
   Syntax: param=<num>
   Description: Parameter controlling the threshold of outlier detection. An outlier is defined as a numerical value that is outside of param multiplied by the inter-quartile range (IQR).
   Default: 2.5

<uselower>
   Syntax: uselower=<bool>
   Description: Controls whether to look for outliers for values below the median in addition to above.
   Default: false
Examples

Example 1: For a timechart of webserver events, transform the outlying average CPU values.

```
404 host="webserver" | timechart avg(cpu_seconds) by host | outlier action=tf
```

Example 2: Remove all outlying numerical values.

```
... | outlier
```

See also

anomalies, anomalousvalue, cluster, kmeans

Finding and removing outliers

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the outlier command.

outputcsv

Description

If you have Splunk Enterprise, this command saves search results to the specified CSV file on the local search head in the $SPLUNK_HOME/var/run/splunk/csv directory. Updates to $SPLUNK_HOME/var/run/*.csv using the outputcsv command are not replicated across the cluster.

If you have Splunk Cloud, you cannot use this command. Instead, you have these options:

- Export search results using REST API. See Export data using the REST APIs in the Search Manual.
Create an alert action that includes a CSV file as an email attachment. See Email notification action in the Alerting Manual.

**Syntax**

outputcsv [append=<bool>] [create_empty=<bool>] [override_if_empty=<bool>] [dispatch=<bool>] [usexml=<bool>] [singlefile=<bool>] [<filename>]

**Optional arguments**

append

**Syntax:** append=<bool>

**Description:** If append is true, the command attempts to append to an existing CSV file, if the file exists. If the CSV file does not exist, a file is created. If there is an existing file that has a CSV header already, the command only emits the fields that are referenced by that header. The command cannot append to .gz files.

**Default:** false

create_empty

**Syntax:** create_empty=<bool>

**Description:** If set to true and there are no results, a zero-length file is created. When set to false and there are no results, no file is created. If the file previously existed, the file is deleted.

**Default:** false

dispatch

**Syntax:** dispatch=<bool>

**Description:** If set to true, refers to a file in the job directory in

$SPLUNK_HOME/var/run/splunk/dispatch/<job id>/.

filename

**Syntax:** <filename>

**Description:** Specify the name of a CSV file to write the search results. This file should be located in $SPLUNK_HOME/var/run/splunk/csv. Directory separators are not permitted in the filename. If no filename is specified, the command rewrites the contents of each result as a CSV row into the _xml field. Otherwise the command writes into a file. The .csv file extension is appended to the filename if the filename has no file extension.

override_if_empty

**Syntax:** override_if_empty=<bool>

479
Description: If `override_if_empty=true` and no results are passed to the output file, the existing output file is deleted. If `override_if_empty=false` and no results are passed to the output file, the command does not delete the existing output file.
Default: true

`singlefile`

**Syntax:** `singlefile=<bool>`

**Description:** If `singlefile` is set to true and the output spans multiple files, collapses it into a single file.

**Default:** true

`usexml`

**Syntax:** `usexml=<bool>`

**Description:** If there is no filename, specifies whether or not to encode the CSV output into XML. This option should not be used when invoking the `outputcsv` from the UI.

Usage

There is no limit to the number of results that can be saved to the CSV file.

**Internal fields and the `outputcsv` command**

When the `outputcsv` command is used there are internal fields that are automatically added to the CSV file. The internal fields that are added to the output in the CSV file are:

- `_raw`
- `_time`
- `_indextime`
- `_serial`
- `_sourcetype`
- `_subsecond`

To exclude internal fields from the output, use the `fields` command and specify the fields that you want to exclude. For example:

```
... | fields - _indextime _sourcetype _subsecond _serial | outputcsv MyTestCsvFile
```
**Multivalued fields**

The `outputcsv` command merges values in a multivalued field into single space-delimited value.

**Distributed deployments**

The `outputcsv` command is not compatible with search head pooling and search head clustering.

The command saves the *.csv file on the local search head in the `$SPLUNK_HOME/var/run/splunk/` directory. The *.csv files are not replicated on the other search heads.

**Examples**

1. **Output search results to a CSV file**

   Output the search results to the `mysearch.csv` file. The CSV file extension is automatically added to the file name if you don't specify the extension in the search.

   ```
   ... | outputcsv mysearch
   ```

2. **Exclude internal fields from the output CSV file**

   You can exclude unwanted internal fields from the output CSV file. In this example, the fields to exclude are `_indextime`, `_sourcetype`, `_subsecond`, and `_serial`.

   ```
   index=_internal sourcetype="splunkd" | head 5 | fields _raw _time | fields - _indextime _sourcetype _subsecond _serial | outputcsv MyTestCsvfile
   ```

3. **Do not delete the CSV file if no search results are returned**

   Output the search results to the `mysearch.csv` file if results are returned from the search. Do not delete the `mysearch.csv` file if no results are returned.

   ```
   ... | outputcsv mysearch.csv override_if_empty=false
   ```

481
See also

inputcsv

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the outputcsv command.

outputlookup

Description

Writes search results to a static lookup table, or KV store collection, that you specify.

Syntax

| outputlookup [append=<bool>] [create_empty=<bool>] [max=<int>] [key_field=<field_name>] [createinapp=<bool>] [override_if_empty=<bool>] (<filename> | <tablename>)

Required arguments

<filename>

Syntax: <string>
Description: The name of the lookup file. The file must end with .csv or .csv.gz.

<tablename>

Syntax: <string>
Description: The name of the lookup table as specified by a stanza name in transforms.conf. The lookup table can be configured for any lookup type (CSV, external, or KV store).

Optional arguments

append

Syntax: append=<bool>
Description: The default, append=false setting, writes the search results to the .csv file or KV store collection. Columns that are not in the current
search results are removed from the file. If set to true, attempts to append search results to an existing .csv file or KV store collection. Otherwise it creates a file. If there is an existing .csv file, the outputlookup command writes only the fields that are present in the previously existing .csv file. An outputlookup search that is run with append=true might result in a situation where the lookup table or collection is only partially updated. This means that a subsequent lookup or inputlookup search on that lookup table or collection might return stale data along with new data. The outputlookup command cannot append to .gz files.

Default: false

create_empty

Syntax: create_empty=<bool>
Description: If set to true and there are no results, a zero-length file is created. When set to false and there are no results, no file is created. If the file previously existed, the file is deleted.
Default: false

createinapp

Syntax: createinapp=<bool>
Description: If set to false, or if there is no current application context, the command creates the file in the system lookups directory.
Default: true

key_field

Syntax: key_field=<field_name>
Description: For KV store-based lookups, uses the specified field name as the key to a value and replaces that value. An outputlookup search using the key_field argument might result in a situation where the lookup table or collection is only partially updated. A subsequent lookup or inputlookup search on that collection might return stale data along with new data. A partial update only occurs with concurrent searches, one with the outputlookup command and a search with the inputlookup command. It is possible that the inputlookup occurs when the outputlookup is still updating some of the records.

max

Syntax: max=<int>
Description: The number of rows to output.
Default: no limit

override_if_empty

Syntax: override_if_empty=<bool>
Description: If `override_if_empty=true` and no results are passed to the output file, the existing output file is deleted. If `override_if_empty=false` and no results are passed to the output file, the command does not delete the existing output file.

Default: true

Usage

The lookup table must be a CSV or GZ file, or a table name specified with a lookup table configuration in `transforms.conf`. The lookup table can refer to a KV store collection or a CSV lookup. The outputlookup command cannot be used with external lookups.

For CSV lookups, if the lookup file does not exist, it is created in the lookups directory of the current application. If the lookup file already exists, it is overwritten with the results of the `outputlookup` command. If the `createinapp` option is set to `false` or if there is no current application context, then the file is created in the system lookups directory.

For permissions in CSV lookups, use the `check_permission` field in `transforms.conf` and `outputlookup_check_permission` in `limits.conf` to restrict write access to users with the appropriate permissions when using the `outputlookup` command. Both `check_permission` and `outputlookup_check_permission` default to `false`. Set to true for Splunk software to verify permission settings for lookups for users. You can change lookup table file permissions in the `.meta` file for each lookup file, or Settings > Lookups > Lookup table files. By default, only users who have the admin or power role can write to a shared CSV lookup file.

For more information about creating lookups, see About lookups in the Knowledge Manager Manual.

For more information about App Key Value Store collections, see About KV store in the Admin Manual.

Appending results

Suppose you have an existing CSV file which contains columns A, D, and J. The results of your search are columns A, C, and J. If you run a search with `outputlookup append=false`, then columns A, C, and J are written to the CSV file. Column D is not retained.
If you run a search with `outputlookup append=true`, then only the columns that are currently in the file are preserved. In this example columns A and J are written to the CSV file. Column C is lost because it does not already exist in the CSV file. Column D is retained.

You can work around this issue by using the `eval` command to add a column to your CSV file before you run the search. For example, if your CSV file is named `foo` you would do something like this:

```
| inputlookup foo | eval c=null | outputlookup foo append=false ....
```

Then run your search and pipe the results to the `fields` command for the columns you want to preserve.

```
... | fields A C J | outputlookup append=true foo
```

**Multivalued fields**

When you output to a static lookup table, the `outputlookup` command merges values in a multivalued field into single space-delimited value. This does not apply to a KV store collection.

**Examples**

**Example 1:** Write to `usertogroup` lookup table as specified in `transforms.conf`.

```
| outputlookup usertogroup
```

**Example 2:** Write to `users.csv` lookup file under

```
$SPLUNK_HOME/etc/system/lookups or $SPLUNK_HOME/etc/apps/*/lookups.
```

```
| outputlookup users.csv
```

**Example 3:** Write to `users.csv` lookup file, if results are returned, under

```
$SPLUNK_HOME/etc/system/lookups or $SPLUNK_HOME/etc/apps/*/lookups. Do not delete the users.csv file if no results are returned.
```

```
| outputlookup users.csv override_if_empty=false
```

**Example 4:** Write food inspection events for Shalimar Restaurant to a KV store collection called `kvstorecoll`. This collection is referenced in a lookup table called `kvstorecoll_lookup`.

```
485
```
Example 5: Write the contents of a CSV file to the KV store collection kvstorecoll using the lookup table kvstorecoll_lookup. This requires usage of both inputlookup and outputlookup.

| inputlookup customers.csv | outputlookup kvstorecoll_lookup |

Example 6: Update field values for a single KV store collection record. This requires usage of inputlookup, outputlookup, and eval. The record is indicated by the value of its internal key ID (the _key field) and is updated with a new customer name and customer city. The record belongs to the KV store collection kvstorecoll, which is accessed through the lookup table kvstorecoll_lookup.

| inputlookup kvstorecoll_lookup | search _key=544948df3ec32d7a4c1d9755 | eval CustName="Marge Simpson" | eval CustCity="Springfield" | outputlookup kvstorecoll_lookup append=True key_field=_key |

To learn how to obtain the internal key ID values of the records in a KV store collection, see Example 5 for the inputlookup command.

See also
inputlookup, lookup, inputcsv, outputcsv

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the outputlookup command.

outputtext

Description

Outputs the raw text (_raw) of results into the _xml field.

The outputtext command was created as an internal mechanism to render event texts for output.

By default, the command xml-escapes the text of events and copies them to the _xml field. If usexml=false, outputtext simply copies the text of events to the
Since `outputtext` is a reporting command, the command will pull all events to the search head and cause the output to show in the statistics UI if used in the web interface.

**Syntax**

`outputtext [usexml=<bool>]`

**Optional arguments**

`usexml`

Syntax: `usexml=<bool>`

Description: If `usexml` is set to true (the default), the copy of the `_raw` field in `_xml` is `xml` escaped. If `usexml` is set to false, the `_xml` field is an exact copy of `_raw`.

**Examples**

*Example 1:*

Output the "_raw" field of your current search into "_xml".

```plaintext
... | outputtext
```

**See also**

`outputcsv`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `outputtext` command.

**overlap**

Note: We do not recommend using the `overlap` command to fill or backfill summary indexes. Splunk Enterprise provides a script called `fill_summary_index.py` that backfills your indexes or fill summary index gaps. If you have Splunk Cloud and need to backfill, open a Support ticket and specify
the time range, app, search name, user and any other details required to enable Splunk Support to backfill the required data. For more information, see "Manage summary index gaps" in the Knowledge Manager Manual.

Description

Find events in a summary index that overlap in time, or find gaps in time during which a scheduled saved search might have missed events.

- If you find a gap, run the search over the period of the gap and summary index the results using "| collect".
- If you find overlapping events, manually delete the overlaps from the summary index by using the search language.

The overlap command invokes an external python script $SPLUNK_HOME/etc/apps/search/bin/sumindexoverlap.py. The script expects input events from the summary index and finds any time overlaps and gaps between events with the same 'info_search_name' but different 'info_search_id'.

Important: Input events are expected to have the following fields: 'info_min_time', 'info_max_time' (inclusive and exclusive, respectively) , 'info_search_id' and 'info_search_name' fields. If the index contains raw events (_raw), the overlap command does not work. Instead, the index should contain events such as chart, stats, and timechart results.

Syntax

overlap

Examples

Example 1:

Find overlapping events in the "summary" index.

index=summary | overlap

See also

collect, sistats, sitop, sirare, sichart, sitimechart
Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the overlap command.

pivot

Description

The pivot command makes simple pivot operations fairly straightforward, but can be pretty complex for more sophisticated pivot operations. Fundamentally this command is a wrapper around the stats and xyseries commands.

The pivot command does not add new behavior, but it might be easier to use if you are already familiar with how Pivot works. See the Pivot Manual. Also, read how to open non-transforming searches in Pivot.

Run pivot searches against a particular data model object. This requires a large number of inputs: the data model, the data model object, and pivot elements.

Syntax

| pivot <datamodel-name> <object-name> <pivot-element>

Required arguments

datamodel-name
Syntax: <string>
Description: The name of the data model to search.

objectname
Syntax: <string>
Description: The name of a data model object to search.

pivot element
Syntax: (<cellvalue>)* (SPLITROW <rowvalue>)* (SPLITCOL colvalue [options])* (FILTER <filter expression>)* (LIMIT <limit expression>)* (ROWSUMMARY <true | false>)* (COLSUMMARY <true | false>)* (SHOWOTHER <true | false>)* (NUMCOLS <num>)* (rowsort [options])*
Description: Use pivot elements to define your pivot table or chart. Pivot elements include cell values, split rows, split columns, filters, limits, row
and column formatting, and row sort options. Cell values always come first. They are followed by split rows and split columns, which can be interleaved, for example: \texttt{avg(val)}, \texttt{SPLITCOL foo}, \texttt{SPLITROW bar}, \texttt{SPLITCOL baz}.

\textbf{Cell value}

\texttt{<cellvalue>}

\textbf{Syntax:} \texttt{<function>(fieldname) [AS <label>]}

\textbf{Description:} Define the values of a cell and optionally rename it. Here, \texttt{label} is the name of the cell in the report.

The set of allowed functions depend on the data type of the \texttt{fieldname}:

- \textbf{Strings:} list, values, first, last, count, and distinct\_count (dc)
- \textbf{Numbers:} sum, count, avg, max, min, stdev, list, and values
- \textbf{Timestamps:} duration, earliest, latest, list, and values
- \textbf{Object or child counts:} count

\textbf{Descriptions for row split-by elements}

\texttt{SPLITROW <rowvalue>}

\textbf{Syntax:} \texttt{SPLITROW <field> [AS <label>] [RANGE start=<value> end=<value> max=<value> size=<value>] [PERIOD (auto | year | month | day | hour | minute | second)] [TRUENAME <label>] [FALSELABEL <label>]}

\textbf{Description:} You can specify one or more of these options on each SPLITROW. The options can appear in any order. You can rename the \texttt{<field>} using "AS <label>", where "label" is the name of the row in the report.

Other options depend on the data type of the \texttt{<field>} specified:

- \textbf{RANGE} applies only for numbers. You do not need to specify all of the options (start, end, max, and size).
- \textbf{PERIOD} applies only for timestamps. Use it to specify the period to bucket by.
- \textbf{TRUENAME} applies only for booleans. Use it to specify the label for true values.
- \textbf{FALSELABEL} applies only for booleans. Use it to specify the label for false values.
**Descriptions for column split-by elements**

SPLITCOL colvalue <options>

**Syntax:**
```
fieldname [ RANGE start=<value> end=<value> max=<value> size=<value> ] [ PERIOD (auto | year | month | day | hour | minute | second) ] [ TRUELABEL <label> ] [ FALSELABEL <label> ]
```

**Description:** You can have none, some, or all of these options on each SPLITCOL. They may appear in any order.

Other options depend on the data type of the field specified (fieldname):

- RANGE applies only for numbers. The options (start, end, max, and size) do not all have to be specified.
- PERIOD applies only for timestamps. Use it to specify the period to bucket by.
- TRUELABEL applies only for booleans. Use it to specify the label for true values.
- FALSELABEL applies only for booleans. Use it to specify the label for false values.

**Descriptions for filter elements**

Filter <filter expression>

**Syntax:**
```
<fieldname> <comparison-operator> <value>
```

**Description:** The expression used to identify values in a field. The comparison operator that you use depends on the type of field value.

- **Strings:** is, contains, in, isNot, doesNotContain, startsWith, endsWith, isNull, isNotNull
  
  For example: ... filter fieldname in (value1, value2, ...)

- **ipv4:** is, contains, isNot, doesNotContain, startsWith, isNull, isNotNull
- **Numbers:** =, !=, <, <=, >, >=, isNull, isNotNull
- **Booleans:** is, isNull, isNotNull

**Descriptions for limit elements**

Limit <limit expression>

**Syntax:**
```
LIMIT <fieldname> BY <limittype> <number> <stats-function>({<fieldname>})
```

**Description:** Use to limit the number of elements in the pivot. The limittype argument specifies where to place the limit. The valid values
are top or bottom. The number argument must be a positive integer. You can use any stats function, such as min, max, avg, and sum.

Example: LIMIT foo BY TOP 10 avg(bar)

Usage

The pivot command is a report-generating command. See Command types.

Generating commands use a leading pipe character and should be the first command in a search.

Examples

Example 1: This command counts the number of events in the "HTTP Requests" object in the "Tutorial" data model.

| pivot Tutorial HTTP_requests count(HTTP_requests) AS "Count of HTTP requests"

This can be formatted as a single value report in the dashboard panel:

![Dashboard Panel]

Example 2: Using the Tutorial data model, create a pivot table for the count of "HTTP Requests" per host.

| pivot Tutorial HTTP_requests count(HTTP_requests) AS "Count" SPLITROW host AS "Server" SORT 100 host

![Pivot Table]

492
See also
datamodel, stats, xyseries

Answers
Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the pivot command.

predict
Description
The predict command forecasts values for one or more sets of time-series data. The command can also fill in missing data in a time-series and provide predictions for the next several time steps.

The predict command provides confidence intervals for all of its estimates. The command adds a predicted value and an upper and lower 95th percentile range to each event in the time-series. See the Usage section in this topic.

Syntax
predict <field-list> [AS <newfield>] [predict_options]

Required arguments
<field-list>
  Syntax: <field>...
  Description: The names of the fields for the variable that you want to predict. You can specify one or more fields.

Optional arguments
<newfield>
  Syntax: <string>
  Description: Renames the fields that are specified in the <field-list>. You do not need to rename every field that you specify in the <field-list>. However, for each field that you want to rename, you must specify a separate AS <newfield> clause.
**Predict options**

**algorithm**

Syntax: algorithm= LL | LLT | LLP | LLP5 | LLB | BiLL

Description: Specify the name of the forecasting algorithm to apply. LL, LLT, LLP, and LLP5 are univariate algorithms. LLB and BiLL are bivariate algorithms. All the algorithms are variations based on the Kalman filter. Each algorithm expects a minimum number of data points. If not enough effective data points are supplied, an error message is displayed. For instance, the field itself might have more than enough data points, but the number of effective data points might be small if the holdback value that you specify is large.

Default: LLP5

<table>
<thead>
<tr>
<th>Algorithm option</th>
<th>Algorithm type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>Local level</td>
<td>A univariate model with no trends and no seasonality. Requires a minimum of 2 data points. The LL algorithm is the simplest algorithm and computes the levels of the time series. For example, each new state equals the previous state, plus the Gaussian noise.</td>
</tr>
<tr>
<td>LLT</td>
<td>Local level trend</td>
<td>A univariate model with trend, but no seasonality. Requires a minimum of 3 data points.</td>
</tr>
<tr>
<td>LLP</td>
<td>Seasonal local level</td>
<td>A univariate model with seasonality. The number of data points must be at least twice the number of periods, using the period attribute. The LLP algorithm takes into account the cyclical regularity of the data, if it exists. If you know the number of periods, specify the period argument. If you do not set the period, this algorithm tries to calculate it.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Algorithm option</th>
<th>Algorithm type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLP5</td>
<td>Combines LLT and LLP models for its prediction.</td>
<td>LLP returns an error message if the data is not periodic.</td>
</tr>
<tr>
<td>LLB</td>
<td>Bivariate local level</td>
<td>If the time series is periodic, LLP5 computes two predictions, one using LLT and the other using LLP. The algorithm then takes a weighted average of the two values and outputs that as the prediction. The confidence interval is also based on a weighted average of the variances of LLT and LLP.</td>
</tr>
<tr>
<td>BiLL</td>
<td>Bivariate local level</td>
<td>A bivariate model that predicts both time series simultaneously. The covariance of the two series is taken into account.</td>
</tr>
</tbody>
</table>

**correlate**

**Syntax:** `correlate=<field>`

**Description:** Specifies the time series that the LLB algorithm uses to predict the other time series. Required when you specify the LLB algorithm. Not used for any other algorithm.

**Default:** None

**future_timespan**

**Syntax:** `future_timespan=<num>`

**Description:** Specifies how many future predictions the predict command will compute. This number must be a non-negative number. You would not use the future_timespan option if `algorithm=LLB`.

**Default:** 5

**holdback**

**Syntax:** `holdback=<num>`

**Description:** Specifies the number of data points from the end that are not to be used by the predict command. Use in conjunction with the
future_timespan argument. For example, 'holdback=10
future_timespan=10' computes the predicted values for the last 10 values
in the data set. You can then judge how accurate the predictions are by
checking whether the actual data point values fall into the predicted
certainty intervals.
Default: 0

lowerXX
Syntax: lower<int>=<field>
Description: Specifies a percentage for the confidence interval and a field
name to use for the lower confidence interval curve. The <int> value is a
percentage that specifies the confidence level. The integer must be a
number between 0 and 100. The <field> value is the field name.
Default: The default confidence interval is 95%. The default field name is
'lower95(prediction(X))' where X is the name of the field to be predicted.

period
Syntax: period=<num>
Description: Specifies the length of the time period, or recurring cycle, in
the time series data. The number must be at least 2. The LLP and LLP5
algorithms attempt to compute the length of time period if no value is
specified. If you specify the span argument with the timechart command,
the unit that you specify for span is the unit used for period. For example,
if your search is ...|timechart span=1d foo2| predict foo2 period=3.
The spans are 1 day and the period for the predict is 3 days. Otherwise,
the unit for the time period is a data point. For example, if there are a
thousand events, then each event is a unit. If you specify period=7, that
means the data recycles after every 7 data points, or events.
Default: None

suppress
Syntax: suppress=<field>
Description: Used with the multivariate algorithms. Specifies one of the
predicted fields to hide from the output. Use suppress when it is difficult to
see all of the predicted visualizations at the same time.
Default: None

upperYY
Syntax: upper<int>=<field>
Description: Specifies a percentage for the confidence interval and a field
name to use for the upper confidence interval curve. The <int> value is a
percentage that specifies the confidence level. This must be a number
between 0 and 100. The <field> value is the field name.
Default: The default confidence interval is 95%. The default field name is 'upper95(prediction(X))' where X is the name of the field to be predicted.

Confidence intervals

The lower and upper confidence interval parameters default to lower95 and upper95. These values specify a confidence interval where 95% of the predictions are expected fall.

It is typical for some of the predictions to fall outside the confidence interval.

- The confidence interval does not cover 100% of the predictions.
- The confidence interval is about a probabilistic expectation and results do not match the expectation exactly.

Usage

Command sequence requirement

The predict command must be preceded by the timechart command. The predict command requires time series data. See the Examples section for more details.

How it works

The predict command models the data by stipulating that there is an unobserved entity which progresses through time in different states.

To predict a value, the command calculates the best estimate of the state by considering all of the data in the past. To compute estimates of the states, the command hypothesizes that the states follow specific linear equations with Gaussian noise components.

Under this hypothesis, the least-squares estimate of the states are calculated efficiently. This calculation is called the Kalman filter, or Kalman-Bucy filter. For each state estimate, a confidence interval is obtained. The estimate is not a point estimate. The estimate is a range of values that contain the observed, or predicted, values.

The measurements might capture only some aspect of the state, but not necessarily the whole state.
**Missing values**

The `predict` command can work with data that has missing values. The command calculates the best estimates of the missing values.

Do not remove events with missing values. Removing the events might distort the periodicity of the data. Do not specify `cont=false` with the `timechart` command. Specifying `cont=false` removes events with missing values.

**Specifying span**

The unit for the `span` specified with the `timechart` command must be seconds or higher. The `predict` command cannot accept subseconds as an input when it calculates the period.

**Examples**

1. **Predict future access**

   This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

   Predict future access based on the previous access numbers that are stored in Apache web access log files. Count the number of access attempts using a span of 1 day.

   ```
sourcetype=access_combined_* | timechart span=1d count(file) as count | predict count
   ```

   The results appear on the Statistics tab. Click the Visualization tab. If necessary change the chart type to a Line Chart.
2. **Predict future purchases for a product**

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range **All time** when you run the search.

Chart the number of purchases made daily for a specific product.

```
sourcetype=access_* action=purchase arcade | timechart span=1d count
```

- This example searches for all purchases events, defined by the `action=purchase` for the arc, and pipes those results into the `timechart` command.
- The `span=1d` argument buckets the count of purchases into daily chunks.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-06-11</td>
<td>17</td>
</tr>
<tr>
<td>2018-06-12</td>
<td>63</td>
</tr>
<tr>
<td>2018-06-13</td>
<td>94</td>
</tr>
<tr>
<td>2018-06-14</td>
<td>82</td>
</tr>
<tr>
<td>2018-06-15</td>
<td>63</td>
</tr>
<tr>
<td>2018-06-16</td>
<td>76</td>
</tr>
<tr>
<td>2018-06-17</td>
<td>70</td>
</tr>
</tbody>
</table>
Add the `predict` command to the search to calculate the prediction for the number of purchases of the Arcade games that might be sold in the near future.

```
sourcetype=access_* action=purchase arcade | timechart span=1d count | predict count
```

The results appear on the Statistics tab. Click the Visualization tab. If necessary change the chart type to a Bar Chart.

### 3. Predict the values using the default algorithm

Predict the values of `foo` using the default LLP5 algorithm, an algorithm that combines the LLP and LLT algorithms.

```
... | timechart span="1m" count AS foo | predict foo
```

### 4. Predict multiple fields using the same algorithm

Predict multiple fields using the same algorithm. The default algorithm in this example.

```
... | timechart ... | predict foo1 foo2 foo3
```

### 5. Specifying different upper and lower confidence intervals

When specifying confidence intervals, the upper and lower confidence interval values do not need to match. This example predicts 10 values for a field using

```
... | timechart span="1m" count AS foo | predict foo1 foo2 foo3
```

500
the LL algorithm, holding back the last 20 values in the data set.

... | timechart span="1m" count AS foo | predict foo AS foobar algorithm=LL upper90=high lower97=low future_timespan=10 holdback=20

6. Predict the values using the LLB algorithm

This example illustrates the LLB algorithm. The foo3 field is predicted by correlating it with the foo2 field.

... | timechart span="1m" count(x) AS foo2 count(y) AS foo3 | predict foo3 AS foobar algorithm=LLB correlate=foo2 holdback=100

7. Omit the last 5 data points and predict 5 future values

In this example, the search abstains from using the last 5 data points and makes 5 future predictions. The predictions correspond to the last 5 values in the data. You can judge how accurate the predictions are by checking whether the observed values fall into the predicted confidence intervals.

... | timechart ... | predict foo holdback=5 future_timespan=5

8. Predict multiple fields using the same algorithm and the same future_timespan and holdback

Predict multiple fields using the same algorithm and same future_timespan and holdback.

... | timechart ... | predict foo1 foo2 foo3 algorithm=LLT future_timespan=15 holdback=5

9. Specify aliases for fields

Use aliases for the fields by specifying the AS keyword for each field.

... | timechart ... | predict foo1 AS foobar1 foo2 AS foobar2 foo3 AS foobar3 algorithm=LLT future_timespan=15 holdback=5

10. Predict multiple fields using different algorithms and options

Predict multiple fields using different algorithms and different options for each field.

... | timechart ... | predict foo1 algorithm=LL future_timespan=15 foo2 algorithm=LLP period=7 future_timespan=7
11. Predict multiple fields using the BiLL algorithm

Predict values for foo1 and foo2 together using the bivariate algorithm BiLL.

... | timechart ... | predict foo1 foo2 algorithm=BiLL
future_timespan=10

See also

trendline, x11

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the predict command.

rangemap

Description

Use the rangemap command to categorize the values in a numeric field. The command adds in a new field called range to each event and displays the category in the range field. The values in the range field are based on the numeric ranges that you specify.

Set the range field to the names of any attribute_name that the value of the input field is within. If no range is matched, the range value is set to the default value.

The ranges that you set can overlap. If you have overlapping values, the range field is created as a multivalue field containing all the values that apply. For example, if low=1-10, elevated=5-15, and the input field value is 10, range=low and code=elevated.

Syntax

rangemap field=<string> (<attribute_name>=<numeric_range>)...
[default=<string>]
**Required arguments**

field
   **Syntax:** field=<string>
   **Description:** The name of the input field. This field must contain numeric values.

**Optional arguments**

attribute_name=numeric_range
   **Syntax:** <string>=<num>-<num>
   **Description:** The <attribute_name> is a string value that is output when the <numeric_range> matches the value in the <field>. The <attribute_name> is a output to the range field. The <numeric_range> is the starting and ending values for the range. The values can be integers or floating point numbers. The first value must be lower than the second. The <numeric_range> can include negative values.
   **Example:** Dislike=-5--1 DontCare=0-0 Like=1-5

default
   **Syntax:** default=<string>
   **Description:** If the input field does not match a range, use this to define a default value.
   **Default:** "None"

**Usage**

The rangemap command is a distributable streaming command. See Command types.

**Examples**

**Example 1:**

Set range to "green" if the date_second is between 1-30; "blue", if between 31-39; "red", if between 40-59; and "gray", if no range matches (for example, if date_second=0).

   ... | rangemap field=date_second green=1-30 blue=31-39 red=40-59 default=gray
**Example 2:**

Sets the value of each event's `range` field to "low" if its `count` field is 0 (zero); "elevated", if between 1-100; "severe", otherwise.

```
... | rangemap field=count low=0-0 elevated=1-100 default=severe
```

**See also**

eval

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the rangemap command.

**rare**

**Description**

Displays the least common values of a field.

Finds the least frequent tuple of values of all fields in the field list. If the `<by-clause>` is specified, this command returns rare tuples of values for each distinct tuple of values of the group-by fields.

This command operates identically to the `top` command, except that the `rare` command finds the least frequent instead of the most frequent.

**Syntax**

```
rare [<top-options>...] <field-list> [by-clause]
```

**Required arguments**

- `<field-list>`
  - **Syntax**: `<string>,...`
  - **Description**: Comma-delimited list of field names.
**Optional arguments**

<top-options>
Syntax: countfield=<string> | limit=<int> | percentfield=<string> | showcount=<bool> | showperc=<bool>
**Description:** Options that specify the type and number of values to display. These are the same <top-options> used by the top command.

(by-clause)
Syntax: BY <field-list>
**Description:** The name of one or more fields to group by.

**Top options**

countfield
Syntax: countfield=<string>
**Description:** The name of a new field to write the value of count into.
**Default:** "count"

limit
Syntax: limit=<int>
**Description:** Specifies how many tuples to return. If you specify limit=0, all values up to maxresultrows are returned. See Limits section. Specifying a value larger than maxresultrows produces an error.
**Default:** 10

percentfield
Syntax: percentfield=<string>
**Description:** Name of a new field to write the value of percentage.
**Default:** "percent"

showcount
Syntax: showcount=<bool>
**Description:** Specify whether to create a field called "count" (see "countfield" option) with the count of that tuple.
**Default:** true

showperc
Syntax: showperc=<bool>
**Description:** Specify whether to create a field called "percent" (see "percentfield" option) with the relative prevalence of that tuple.
**Default:** true
Usage

The `rare` command is a **transforming command**. See Command types.

The number of results returned by the `rare` command is controlled by the `limit` argument. The default value for the `limit` argument is 10. You can change this limit up to the maximum value specified in the `maxresultrows` setting in the `[rare]` stanza in the limits.conf file. The default maximum is 50,000, which effectively keeps a ceiling on the memory that the `rare` command uses.

Examples

1. **Return the least common values in a field**

   Return the least common values in the "url" field. Limits the number of values returned to 5.

   ```
   ... | rare url limit=5
   ```

2. **Return the least common values organized by host**

   Find the least common values in the "user" field for each "host" value. By default, a maximum of 10 results are returned.

   ```
   ... | rare user by host
   ```

See also

top, stats, sirare

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the rare command.

redistribute

Description

The `redistribute` command implements **parallel reduce** search processing to shorten the search runtime of a set of supported SPL commands. Apply the
The `redistribute` command requires a **distributed search** environment where indexers have been configured to operate as **intermediate reducers**. Only users with **roles** that have the `run_multi_phased_searches` capability can use the `redistribute` command in searches.

You can use the `redistribute` command only once in a search.

**Syntax**

```
redistribute [num_ofReducers=<int>] [by-clause]
```

**Required arguments**

None.

**Optional arguments**

`num_ofReducers`

- **Syntax:** `num_ofReducers=<int>`
- **Description:** Specifies the number of indexers in the indexer pool that are repurposed as intermediate reducers.
- **Default:** The default value for `num_ofReducers` is controlled by three settings in the `limits.conf` file: `maxReducersPerPhase`, `winningRate`, and `reducers`. If these settings are not changed, by default the Splunk software sets `num_ofReducers` to 50 percent of your indexer pool, with a maximum of 4 indexers. See **Usage** for more information.

`by-clause`

- **Syntax:** `BY <field-list>`
- **Description:** The name of one or more fields to group by. You cannot use a wildcard character to specify multiple fields with similar names. You must specify each field separately. See **Using the by-clause** for more information.

**Usage**

In Splunk deployments that have distributed search, a two-phase map-reduce process is typically used to determine the final result set for the search. Search results are mapped at the indexer layer and then reduced at the search head.
The `redistribute` command inserts an intermediary reduce phase to the map-reduce process, making it a three-phase map-reduce-reduce process. This three-phase process is parallel reduce search processing.

In the intermediary reduce phase, a subset of the indexers become intermediate reducers. The intermediate reducers perform reduce operations for the search commands and then pass the results on to the search head, where the final result reduction and aggregation operations are performed. This parallelization of reduction work that otherwise would be done entirely by the search head can result in faster completion times for high-cardinality searches that aggregate large numbers of search results.

For information about managing parallel reduce processing at the indexer level, including configuring indexers to operate as intermediate reducers, see Overview of parallel reduce search processing, in the *Distributed Search* manual.

If you use Splunk Cloud, use `redistribute` only when your indexers are operating with a low to medium average load. You do not need to perform any configuration tasks to use the `redistribute` command.

**Supported commands**

The `redistribute` command supports only *streaming commands* and the following nonstreaming commands:

- `stats`
- `tstats`
- `streamstats`
- `eventstats`
- `sichart`
- `sitimechart`.

The `redistribute` command also supports the `transaction` command, when the `transaction` command is operating on only one field. For example, the `redistribute` command cannot support the `transaction` command when the following conditions are true:

- The `redistribute` command has multiple fields in its `<by-clause>` argument.
- The `transaction` command has multiple fields in its `<field-list>` argument.
- You use the `transaction` command in a mode where no field is specified.
For best performance, place `redistribute` immediately before the first supported nonstreaming command that has high-cardinality input.

**When search processing moves to the search head**

The `redistribute` command moves the processing of a search string from the intermediate reducers to the search head in the following circumstances:

- It encounters a nonstreaming command that it does not support.
- It encounters a command that it supports but that does not include a split-by field.
- It encounters a command that it supports and that includes split-by fields, but the split-by fields are not a superset of the fields that are specified in the by-clause argument of the `redistribute` command.
- It detects that a command modifies values of the fields specified in the by-clause of the `redistribute` command.

**Using the by-clause to determine how results are partitioned on the reducers**

At the start of the intermediate reduce phase, the `redistribute` command takes the mapped search results and redistributes them into partitions on the intermediate reducers according to the fields specified by the by-clause argument. If you do not specify any by-clause fields, the search processor uses the field or fields that work best with the commands that follow the `redistribute` command in the search string.

**Command type**

The `redistribute` command is an orchestrating command, which means that it controls how a search runs. It does not focus on the events processed by the search. The `redistribute` command instructs the distributed search query planner to convert centralized streaming data into distributed streaming data by distributing it across the intermediate reducers.

For more information about command types, see Types of commands in the Search Manual.

**Setting the default number of intermediate reducers**

The default value for the `num_ofReducers` argument is controlled by three settings in the limits.conf file: `maxReducersPerPhase`, `winningRate`, and `reducers`. 509
<table>
<thead>
<tr>
<th>Setting name</th>
<th>Definition</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxReducersPerPhase</td>
<td>The maximum number of indexers that can be used as intermediate reducers in the intermediate reduce phase.</td>
<td>4</td>
</tr>
<tr>
<td>winningRate</td>
<td>The percentage of indexers that can be selected from the total pool of indexers and used as intermediate reducers in a parallel reduce search process. This setting applies only when the reducers setting is not configured.</td>
<td>50</td>
</tr>
<tr>
<td>reducers</td>
<td>A list of valid indexers that are to be used as dedicated intermediate reducers for parallel reduce search processing. When you run a search with the redistribute command, the valid indexers in the reducers list are the only indexers that are used for parallel reduce operations. If the number of valid indexers in the reducers list exceeds the maxReducersPerPhase value, the Splunk platform randomly selects a set of indexers from the reducers list that meets the maxReducersPerPhase limit.</td>
<td>&quot; &quot; (empty list)</td>
</tr>
</tbody>
</table>

If you decide to add 7 of your indexers to the reducers list, the winningRate setting ceases to be applied, and the num_of_reducers argument defaults to 4 indexers. The Splunk platform randomly selects four indexers from the reducers list to act as intermediate reducers each time you run a valid redistribute search.

If you provide a value for the num_of_reducers argument that exceeds the limit set by the maxReducersPerPhase setting, the Splunk platform sets the number of reducers to the maxReducersPerPhase value.

**The redistribute command and search head data**

Searches that use the redistribute command ignore all data on the search head. If you plan to use the redistribute command, the best practice is to forward all search head data to the indexer layer. See Best Practice: Forward search head data to the indexer layer in the *Distributed Search* manual.
Using the redistribute command in chart and timechart searches

If you want to add the redistribute command to a search that uses the chart or timechart commands to produce statistical results that can be used for chart visualizations, include either the sichart command or the sitimechart command in the search as well. The redistribute command uses these si- commands to perform the statistical calculations for the reporting commands on the intermediate reducers. When the redistribute command moves the results to the search head, the chart or timechart command transforms the results into a format that can be used for chart visualizations.

A best practice is to use the same syntax and values for both commands. For example, if you want to have | timechart count by referrer_domain in your redistribute search, insert | sitimechart count by referrer_domain into the search string:

```
index=main | redistribute | transaction referrer_domain | search eventcount>500 | sitimechart count by referrer_domain | search referrer_domain=*.net | timechart count by referrer_domain
```

If an order-sensitive command is present in the search

Certain commands that the redistribute command supports explicitly return results in a sorted order. As a result of the partitioning that takes place when the redistribute command is run, the Splunk platform loses the sorting order. If the Splunk platform detects that an order-sensitive command, such as streamstats, is used in a redistribute search, it automatically inserts sort into the search as it processes it.

For example, the following search includes the streamstats command, which is order-sensitive:

```
... | redistribute by host | stats count by host | streamstats count by host, source
```

The Splunk platform adds a sort segment before the streamstats segment when it processes the search. You can see the sort segment in the search string if you inspect the search job after you run it.

```
... | redistribute by host | stats count by host | sort 0 str(host) | streamstats count by host, source
```

The stats and streamstats segments are processed on the intermediate reducers because they both split by the host field, the same field that the
The \texttt{redistribute} command is distributing on. The work of the \texttt{sort} segment is split between the indexers during the map phase of the search and the search head during the final reduce phase of the search.

\textbf{If you require sorted results from a redistribute search}

If you require the results of a \texttt{redistribute} search to be sorted in that exact order, use \texttt{sort} to perform the sorting at the search head. There is an additional performance cost to event sorting after the \texttt{redistribute} command partitions events on the intermediate reducers.

The following search provides ordered results:

\begin{verbatim}
search * | stats count by foo
\end{verbatim}

If you want to get that same event ordering while also adding \texttt{redistribute} to the search to speed it up, add \texttt{sort} to the search:

\begin{verbatim}
search * | redistribute | stats count by foo | sort 0 str(foo)
\end{verbatim}

The \texttt{stats} segment of this search is processed on the intermediate reducers. The work of the \texttt{sort} segment is split between the indexers during the map phase of the search and the search head during the final reduce phase of the search.

\textbf{Redistribute and virtual indexes}

The \texttt{redistribute} command does not support searches of virtual indexes. The \texttt{redistribute} command also does not support unified searches if their time ranges are long enough that they run across virtual archive indexes. For more information, see the following \textit{Splunk Analytics for Hadoop} topics:

\begin{itemize}
  \item About virtual indexes
  \item Configure and run unified search
\end{itemize}

\textbf{Examples}

\textit{1. Speed up a search on a large high-cardinality dataset}

In this example, the \texttt{redistribute} command is applied to a \texttt{stats} search that is running over an extremely large high-cardinality dataset. The \texttt{redistribute} command reduces the completion time for the search.
The intermediate reducers process the portion of the search in parallel, lowering the completion time for the search. The search head aggregates the results.

2. Speed up a timechart search without declaring a by-clause field to redistribute on

This example uses a search over an extremely large high-cardinality dataset. The search string includes the `eventstats` command, and it uses the `sitimechart` command to perform the statistical calculations for a timechart operation. The search uses the `redistribute` command to reduce the completion time for the search. A `by-clause` field is not specified, so the search processor selects one.

When this search runs, the intermediate reducers process the `eventstats` and `sitimechart` segments of the search in parallel, reducing the overall completion time of the search. On the search head, the `timechart` command takes the reduced `sitimechart` calculations and transforms them into a format that can be used for charts and visualizations.

Because a `by-clause` field is not identified in the search string, the intermediate reducers redistribute and partition events on the `source` field.

3. Speed up a search that uses `tstats` to generate events

This example uses a search over an extremely large high-cardinality dataset. This search uses the `tstats` command in conjunction with the `sitimechart` and `timechart` commands. The `redistribute` command reduces the completion time for the search.

You have to place the `tstats` command at the start of the search string with a leading pipe character. When you use the `redistribute` command in conjunction with `tstats`, you must place the `redistribute` command after the `tstats` segment of the search.
In this example, the `tstats` command uses the `prestats=t` argument to work with the `sitimechart` and `timechart` commands.

The `redistribute` command causes the intermediate reducers to process the `sitimechart` segment of the search in parallel, reducing the overall completion time for the search. The reducers then push the results to the search head, where the `timechart` command processes them into a format that you can use for charts and visualizations.

4. Speed up a search that includes a mix of supported and unsupported commands

This example uses a search over an extremely large high-cardinality dataset. The search uses the `redistribute` command to reduce the search completion time. The search includes commands that are both supported and unsupported by the `redistribute` command. It uses the `sort` command to sort of the results after the rest of the search has been processed. You need the `sort` command for event sorting because the `redistribute` process undoes the sorting naturally provided by commands in the `stats` command family.

```
... | redistribute | eventstats count by user, source | where count >10
| sort 0 -num(count)
```

In this example, the intermediate reducers process the `eventstats` and `where` segments in parallel. Those portions of the search complete faster than they would when the `redistribute` command is not used.

The Splunk platform divides the work of processing the `sort` portion of the search between the indexer and the search head.

5. Speed up a search where a supported command splits by fields that are not in the redistribute command by-clause argument

In this example, the `redistribute` command redistributes events across the intermediate reducers by the `source` field. The search includes two commands that are supported by the `redistribute` command but only one of them is processed on the intermediate reducers.

```
... | redistribute by source | eventstats count by source, host | where count > 10 | stats count by userid, host
```

In this case, the `eventstats` segment of the search is processed in parallel by the intermediate reducers because it includes `source` as a split-by field. The `where` segment is also processed on the intermediate reducers.
The *stats* portion of the search, however, is processed on the search head because its split-by fields are not a superset of the set of fields that the events have been redistributed by. In other words, the *stats* split-by fields do not include *source*.

**regex**

**Description**

The regex command removes results that do not match the specified regular expression.

**Syntax**

```
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 email="/^[a-z0-9_.-]+@[a-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 to start at 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 () character is used to escape the dot ( . ) character. The dot character is escaped, because a non-escaped dot matches any character. The plus ( + ) sign specifies to match from 1 to unlimited characters in this group. In this example this part of the expression matches <strong>buttercup</strong> in the email address <a href="mailto:buttercup@example.com">buttercup@example.com</a>.</td>
</tr>
<tr>
<td><code>@</code></td>
<td>Matches the at symbol.</td>
</tr>
<tr>
<td>Part of the expression</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>([\da-z.-]+)</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 (+) 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>([a-z.]{2,6})</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>
</tbody>
</table>

See also

`rex`, `search`

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the regex command.

_relevancy_

Description

Calculates how well the event matches the query based on how well the events `_raw` field matches the keywords of the `search`. Saves the result into a field named "relevancy". Useful for retrieving the best matching events/documents, rather than the default time-based ordering. Events score a higher relevancy if they have more rare search keywords, more frequently, in fewer terms. For example a search for `disk error` will favor a short event/document that has 'disk' (a rare term) several times and 'error' once, than a very large event that has 'disk' once and 'error' several times.

**Note:** The relevancy command does not currently work. See SPL-93039 on the Known issues page here:
Syntax
relevancy

Examples

Example 1: Calculate the relevancy of the search and sort the results in descending order.

disk error | relevancy | sort -relevancy

See also

abstract, highlight, sort

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the relevancy command.

reltime

Description

Creates a relative time field, called 'reltime', and sets this field to a human readable value of the difference between 'now' and '_time'. Human-readable values look like "5 days ago", "1 minute ago", "2 years ago", and so on.

Syntax
reltime

Usage

The reltime command is a distributable streaming command. See Command types.
Examples

Example 1:

Adds a field called `reltime` to the events returned from the search.

```
... | reltime
```

See also

`convert`

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `reltime` command.

rename

Description

Use the `rename` command to rename one or more fields. This command is useful for giving fields more meaningful names, such as "Product ID" instead of "pid". If you want to rename fields with similar names, you can use a wildcard character. See the Usage section.

Syntax

```
rename <wc-field> AS <wc-field>...
```

Required arguments

`wc-field`

**Syntax**: `<string>`

**Description**: The name of a field and the name to replace it. You can use wild card characters in the field names. Names with spaces must be enclosed in quotation marks.
**Usage**

The *rename* command is a distributable streaming command. See Command types.

**Rename with a phrase**

Use quotation marks when you rename a field with a phrase.

```
... | rename SESSIONID AS "The session ID"
```

**Rename multiple, similarly named fields**

Use wildcards to rename multiple fields.

```
... | rename *ip AS *IPaddress
```

If both the source and destination fields are wildcard expressions with the same number of wildcards, the renaming will carry over the wildcarded portions to the destination expression. See Examples.

**You cannot rename one field with multiple names**

You cannot rename one field with multiple names. For example if you have field A, you cannot specify `| rename A as B, A as C`. This rule also applies to other commands where you can rename fields, such as the *stats* command.

The following example is not valid.

```
... | stats first(host) AS site, first(host) AS report
```

**You cannot merge multiple fields into one field**

You cannot use the *rename* command to merge multiple fields into one field because null, or non-present, fields are brought along with the values.

For example, if you have events with either *product_id* or *pid* fields, `... | rename pid AS product_id` would not merge the *pid* values into the *product_id* field. It overwrites *product_id* with Null values where *pid* does not exist for the event. See the *eval* command and *coalesce()* function.
**Renaming a field that does not exist**

Renaming a field can cause loss of data.

Suppose you rename fieldA to fieldB, but fieldA does not exist.

- If fieldB does not exist, nothing happens.
- If fieldB does exist, the result of the rename is that the data in fieldB is removed. The data in fieldB will contain null values.

**Examples**

**Example 1:**

Rename the "_ip" field to "IPAddress".

```
... | rename _ip AS IPAddress
```

**Example 2:**

Rename fields beginning with "foo" to begin with "bar".

```
... | rename foo* AS bar*
```

**Example 3:**

Rename the "count" field. Names with spaces must be enclosed in quotation marks.

```
... | rename count AS "Count of Events"
```

**See also**

fields, replace, table

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the rename command.
replace

Description

Replaces field values with the values that you specify.

Replaces a single occurrence of the first string with another string in the specified fields. If you do not specify one or more fields, the value is replaced in all fields.

Syntax

replace (<wc-string> WITH <wc-string>)... [IN <field-list>]

Required arguments

wc-string

Syntax: <string>

Description: Specify one or more field values and their replacements.

You can use wildcard characters to match one or multiple terms.

Optional arguments

field-list

Syntax: <string> ...

Description: Specify a comma or space delimited list of one or more field names for the field value replacements. To replace values on _internal fields, you must specify the field name with the IN <fieldname> clause.

Usage

The replace command is a distributable streaming command. See Command types.

Non-wildcard replacement values specified later take precedence over those replacements specified earlier. For a wildcard replacement, fuller matches take precedence over lesser matches. To assure precedence relationships, you are advised to split the replace into two separate invocations. When using wildcard replacements, the result must have the same number of wildcards, or none at all. Wildcards ( * ) can be used to specify many values to replace, or replace values with.
Examples

1. Replace a value in all fields

Change any host value that ends with "localhost" to simply "localhost" in all fields.

... | replace *localhost WITH localhost

2. Replace a value in a specific field

Replace an IP address with a more descriptive name in the host field.

... | replace 127.0.0.1 WITH localhost IN host

3. Change the value of two fields

Replaces the values in the start_month and end_month fields. You can separate the names in the field list with spaces or commas.

... | replace aug WITH August IN start_month end_month

4. Change the order of values in a field

In the host field, change the order of string values that contain the word localhost so that the string "localhost" precedes the other strings.

... | replace "* localhost" WITH "localhost *" IN host

5. Replace multiple values in a field

Replace the values in a field with more descriptive names. Separate the value replacements with comma.

... | replace 0 WITH Critical, 1 WITH Error IN msg_level

6. Replace empty strings

Search for an error message and replace empty strings with a whitespace.

This example will not work unless you have values that are actually the empty string, which is not the same as not having a value.

"Error exporting to XYZ :" | rex "Error exporting to XYZ:(?.*):" | replace "" WITH " " IN errmsg

523
7: Replace values in an internal field

Replace values of the internal field _time.

```
sourcetype=* | head 5 | eval _time="XYZ" | stats count BY _time | replace *XYZ* WITH *ALL* IN _time
```

See also

fillnull, rename

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the replace command.

rest

Description

The `rest` command reads a Splunk REST API endpoint and returns the resource data as a search result.

For information about the REST API, see the REST API User Manual.

Syntax

```
| rest <rest-uri> [count=<int>] [splunk_server=<wc-string>] [splunk_server_group=<wc-string>]... [timeout=<int>] [<get-arg-name>=<get-arg-value>]...
```

Required arguments

rest-uri

Syntax: <uri>

Description: URI path to the Splunk REST API endpoint.

Optional arguments

count

Syntax: count=<int>
**Description:** Limits the number of results returned from each REST call. For example, you have four indexers and one search head. You set the limit to `count=25000`. This results in a total limit of 125000, which is 25000 x 5.
When `count=0`, there is no limit.
**Default:** 0

`get-arg-name`

**Syntax:** `<string>`

**Description:** REST argument name.

`get-arg-value`

**Syntax:** `<string>`

**Description:** REST argument value.

`splunk_server`

**Syntax:** `splunk_server=<wc-string>`

**Description:** Specifies the distributed search peer from which to return results. You can specify only one `splunk_server` argument. However, you can use a wildcard character when you specify the server name to indicate multiple servers. For example, you can specify `splunk_server=peer01` or `splunk_server=peer*`. Use `local` to refer to the search head.
**Default:** All configured search peers return information

`splunk_server_group`

**Syntax:** `splunk_server_group=<wc-string>...`

**Description:** Limits the results to one or more server groups. You can specify a wildcard character in the string to indicate multiple server groups.

`timeout`

**Syntax:** `timeout=<int>`

**Description:** Specify the timeout in seconds when waiting for the REST endpoint to respond.
**Default:** 60

**Usage**

The `rest` command authenticates using the ID of the person that runs the command.
Examples

Example 1: Access saved search jobs.

| rest /services/search/jobs count=0 splunk_server=local | search isSaved=1

Example 2: Find all saved searches with search strings that include the speccsv sourcetype.

| rest /services/saved/searches splunk_server=local | rename search AS saved_search | fields author, title, saved_search | search saved_search=*speccsv*

Example 3: Add current search user to all events (useful for creating reports that only show events associated with logged in user).

* | head 10 | join [ | rest splunk_server=local /services/authentication/current-context | rename username as auth_user_id | fields auth_user_id ]

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the rest command.

return

Description

Returns values from a subsearch.

The `return` command is used to pass values up from a subsearch. The command replaces the incoming events with one event, with one attribute: "search". To improve performance, the `return` command automatically limits the number of incoming results with the `head` command and the resulting fields with the `fields` command.

By default, the `return` command uses only the first row of results. Use the `count` argument to specify the number of results to use.
Syntax

return [<count>] [<alias>=<field>...] [<field>...] [$<field>...]  

Required arguments

None.

Optional arguments

<count>

Syntax: <int>
Description: Specify the number of rows.
Default: 1, which is the first row of results passed into the command.

(alias)

Syntax: <alias>=<field>...
Description: Specify the field alias and value to return. You can specify multiple pairs of aliases and values, separated by spaces.

(field)

Syntax: <field>...
Description: Specify one or more fields to return, separated by spaces.

($field)

Syntax: <$field>
Description: Specify one or more field values to return, separated by spaces.

Usage

The command is convenient for outputting a field name, a alias-value pair, or just a field value.

<table>
<thead>
<tr>
<th>Output</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field name</td>
<td>return source</td>
</tr>
<tr>
<td>Alias=value</td>
<td>return ip=srcip</td>
</tr>
<tr>
<td>Value</td>
<td>return $srcip</td>
</tr>
</tbody>
</table>

By default, the return command uses only the first row of results. You can specify multiple rows, for example 'return 2 ip'. Each row is viewed as an OR clause, that is, output might be '(ip=10.1.11.2) OR (ip=10.2.12.3)'. Multiple
values can be specified and are placed within OR clauses. So, 'return 2 user ip' might output 'user=bob ip=10.1.11.2) OR (user=fred ip=10.2.12.3)'.

In most cases, using the return command at the end of a subsearch removes the need for head, fields, rename, format, and dedup.

**Duplicate values**

Suppose you have the following search:

```
sourceType=WinEventLog:Security | return 2 user
```

You might logically expect the command to return the first two distinct users. Instead the command looks at the first two events, based on the ordering from the implied head command. The return command returns the users within those two events. The command does not determine if the user value is unique. If the same user is listed in these events, the command returns only the one user.

To return unique values, you need to include the dedup command in your search. For example:

```
sourceType=WinEventLog:Security | dedup user | return 2 user
```

**Quotations in returned fields**

The return command does not escape quotation marks that are in the fields that are returned. You must use an eval command to escape the quotation marks before you use the return command. For example:

```
...[search eval field2=replace(field1,"\"","\\\") | return field2]
```

**Examples**

**Example 1:**

Search for 'error ip=<someip>', where <someip> is the most recent ip used by user 'boss'.

```
error [ search user=boss | return ip ]
```
**Example 2:**

Search for `error (user=user1 ip=ip1) OR (user=user2 ip=ip2)`, where the users and IPs come from the two most-recent logins.

```bash
error [ search login | return 2 user ip ]
```

**Example 3:**

Return to eval the userid of the last user, and increment it by 1.

```bash
... | eval nextid = 1 + [ search user=* | return $id ] | ...
```

**See also**

format, search

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the return command.

**reverse**

**Syntax**

```bash
reverse
```

**Description**

Reverses the order of the results.

**Note:** the reverse command does not affect which events are returned by the search, only the order in which they are displayed. For the CLI, this includes any default or explicit maxout setting.

**Note:** Reverse on very large result sets, which means sets with millions of results or more, requires large amounts of temporary storage, I/O, and time.
Examples

**Example 1:**

Reverse the order of a result set.

```shell
... | reverse
```

See also

head, sort, tail

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the reverse command.

**rex**

Description

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

The `rex` command matches the value of the specified field against the unanchored regular expression and extracts the named groups into fields of the corresponding names.

When `mode=sed`, the given sed expression used to replace or substitute characters is applied to the value of the chosen field. This sed-syntax is also used to mask sensitive data at **index-time**. Read about using sed to anonymize data in the *Getting Data In* Manual.

If a field is not specified, the regular expression or sed expression is applied to the `_raw` field. Running the `rex` command against the `_raw` field might have a performance impact.

Use the `rex` command for **search-time field extraction** or string replacement and character substitution.
Syntax

rex [field=<field>] ( <regex-expression> [max_match=<int>] [offset_field=<string>] ) | (mode=sed <sed-expression>)

Required arguments

You must specify either <regex-expression> or mode=sed <sed-expression>.

regex-expression

Syntax: "<string>"
Description: The PCRE regular expression that defines the information to match and extract from the specified field. Quotation marks are required.

mode

Syntax: mode=sed
Description: Specify to indicate that you are using a sed (UNIX stream editor) expression.

sed-expression

Syntax: "<string>"
Description: When mode=sed, specify whether to replace strings (s) or substitute characters (y) in the matching regular expression. No other sed commands are implemented. Quotation marks are required. Sed mode supports the following flags: global (g) and Nth occurrence (N), where N is a number that is the character location in the string.

Optional arguments

field

Syntax: field=<field>
Description: The field that you want to extract information from.
Default: _raw

max_match

Syntax: max_match=<int>
Description: Controls the number of times the regex is matched. If greater than 1, the resulting fields are multivalued fields.
Default: 1, use 0 to mean unlimited.

offset_field

Syntax: offset_field=<string>
Description: If provided, a field is created with the name specified by `<string>`. This value of the field has the endpoints of the match in terms of zero-offset characters into the matched field. For example, if the rex expression is "(?<tenchars>.{10})", this matches the first ten characters of the field, and the offset_field contents is "0-9".

Default: unset

Sed expression

When using the rex command in sed mode, you have two options: replace (s) or character substitution (y).

The syntax for using sed to replace (s) text in your data is: "s/<regex>/<replacement>/<flags>"

- `<regex>` is a PCRE regular expression, which can include capturing groups.
- `<replacement>` is a string to replace the regex match. Use \n for backreferences, where "n" is a single digit.
- `<flags>` can be either: g to replace all matches, or a number to replace a specified match.

The syntax for using sed to substitute characters is: "y/<string1>/<string2>/"

- This substitutes the characters that match `<string1>` with the characters in `<string2>`.

Usage

The rex command is a distributable streaming command. See Command types.

Use the rex command to either extract fields using regular expression named groups, or replace or substitute characters in a field using sed expressions. Use the regex command to remove results that do not match the specified regular expression.

Splunk SPL uses perl-compatible regular expressions (PCRE).

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 Splunk Enterprise regular expressions in the Knowledge Manager Manual.

Examples

1. Extract email values using regular expressions

Extract email values from events to create from and to fields in your events. For example, you have events such as:

```
Mon Mar 19 20:16:27 2018 Info: Bounced: DCID 8413617 MID 19338947 From: <MariaDubois@example.com> To: <zecora@buttercupgames.com> RID 0 - 5.4.7 - Delivery expired (message too old) ('000', ['timeout'])

Mon Mar 19 20:16:03 2018 Info: Delayed: DCID 8414309 MID 19410908 From: <WeiZhang@example.com> To: <mcintosh@buttercupgames.com> RID 0 - 4.3.2 - Not accepting messages at this time ('421', ['4.3.2 try again later'])

Mon Mar 19 20:16:02 2018 Info: Bounced: DCID 0 MID 19408690 From: <Exit_Desk@sample.net> To: <lyra@buttercupgames.com> RID 0 - 5.1.2 - Bad destination host ('000', ['DNS Hard Error looking up mahidnrasatyambsg.com (MX): NXDomain'])

Mon Mar 19 20:15:53 2018 Info: Delayed: DCID 8414166 MID 19410657 From: <Manish_Das@example.com> To: <dash@buttercupgames.com> RID 0 - 4.3.2 - Not accepting messages at this time ('421', ['4.3.2 try again later'])

When the events were indexed, the From and To values were not identified as fields. You can use the rex command to extract the field values and create from and to fields in your search results.

The from and to lines in the _raw events follow an identical pattern. Each from line is From: and each to line is To:. The email addresses are enclosed in angle brackets. You can use this pattern to create a regular expression to extract the values and create the fields.

```
source="cisco_esa.txt" | rex field=_raw "From: <(?<from>.*>) To: <(?<to>.*>)"
```

You can remove duplicate values and return only the list of address by adding the dedup and table commands to the search.

```
source="cisco_esa.txt" | rex field=_raw "From: <(?<from>.*>) To: <(?<to>.*>)" | dedup from to | table from to
```

533
The results look something like this:

![New Search](image)

**Example 2:**

Extract "user", "app" and "SavedSearchName" from a field called "savedsearch_id" in scheduler.log events. If `savedsearch_id=bob;search;my_saved_search` then user=bob, app=search and SavedSearchName=my_saved_search

```plaintext
... | rex field=savedsearch_id "(?<user>\w+);(?<app>\w+);(?<SavedSearchName>\w+)"
```

**Example 3:**

Use `sed` syntax to match the regex to a series of numbers and replace them with an anonymized string.

```plaintext
... | rex field=ccnumber mode=sed "s/(\d{4}-){3}/XXXX-XXXX-XXXX-/g"
```

**Example 4:**

Display IP address and ports of potential attackers.

```plaintext
sourcetype=linux_secure port "failed password" | rex "\s+(?<ports>port \d+)|top src_ip ports showperc=0
```

This search used `rex` to extract the port field and values. Then, it displays a table of the top source IP addresses (src_ip) and ports the returned with the search for potential attackers.

**See also**

extract, kvform, multikv, regex, spath, xmlkv
rtorder

Description

Buffers events from real-time search to emit them in ascending time order when possible.

The rtorder command creates a streaming event buffer that takes input events, stores them in the buffer in ascending time order, and emits them in that order from the buffer. This is only done after the current time reaches at least the span of time given by buffer_span, after the timestamp of the event.

Events are also emitted from the buffer if the maximum size of the buffer is exceeded.

If an event is received as input that is earlier than an event that has already been emitted previously, the out of order event is emitted immediately unless the discard option is set to true. When discard is set to true, out of order events are always discarded to assure that the output is strictly in time ascending order.

Syntax

rtorder [discard=<bool>] [buffer_span=<span-length>] [max_buffer_size=<int>]

Optional arguments

buffer_span
  Syntax: buffer_span=<span-length>
  Description: Specify the length of the buffer.
  Default: 10 seconds

discard
  Syntax: discard=<bool>
  Description: Specifies whether or not to always discard out-of-order events.
  Default: false

max_buffer_size
  Syntax: max_buffer_size=<int>
  Description: Specifies the maximum size of the buffer.
  Default: 50000, or the max_result_rows setting of the [search] stanza in limits.conf.
Examples

**Example 1:**

Keep a buffer of the last 5 minutes of events, emitting events in ascending time order once they are more than 5 minutes old. Newly received events that are older than 5 minutes are discarded if an event after that time has already been emitted.

```plaintext
... | rtorder discard=t buffer_span=5m
```

See also

sort

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the rtorder command.

**run**

The `run` command is an alias for the `script` command. See the `script` command for the syntax and examples.

**savedsearch**

**Description**

Runs a saved search, or report, and returns the search results of a saved search. If the search contains replacement placeholder terms, such as `$replace_me$`, the search processor replaces the placeholders with the strings you specify. For example:

```plaintext
|savedsearch mysearch replace_me="value"
```

**Syntax**

```plaintext
| savedsearch <savedsearch_name> [<savedsearch-options>...]
```
Required arguments

savedsearch_name
   Syntax: <string>
   Description: Name of the saved search to run.

Optional arguments

savedsearch-options
   Syntax: <substitution-control> | <replacement>
   Description: Specify whether substitutions are allowed. If allowed, specify
   the key-value pair to use in the string substitution replacement.

   substitution-control
      Syntax: nosubstitution=<bool>
      Description: If true, no string substitution replacements are made.
      Default: false

   replacement
      Syntax: <field>=<string>
      Description: A key-value pair to use in string substitution
      replacement.

Usage

The savedsearch command is a generating command and must start with a
leading pipe character.

The savedsearch command always runs a new search. To reanimate the results
of a previously run search, use the loadjob command.

Time ranges

• If you specify All Time in the time range picker, the savedsearch
  command uses the time range that was saved with the saved search.

• If you specify any other time in the time range picker, the time range that
  you specify overrides the time range that was saved with the saved search.
Examples

Example 1

Run the saved search "mysecurityquery".

| savedsearch mysecurityquery

Example 2

Run the saved search "mysearch". Where the replacement placeholder term $replace_me$ appears in the saved search, use "value" instead.

| savedsearch mysearch replace_me="value"...

See also

search, loadjob

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the savedsearch command.

script

Description

Calls an external python program that can modify or generate search results. Scripts must be declared in the commands.conf file and be located in the $SPLUNK_HOME/etc/apps/<app_name>/bin/ directory. The script is executed using $SPLUNK_HOME/bin/python.

If you are using Splunk Cloud and want to install a custom script, file a Support ticket. Before being installed, your script is checked to ensure it complies with Splunk requirements for security, data safety, and so on.

Syntax

script <script-name> [<script-arg>...] [maxinputs=<int>]

538
**Required arguments**

script-name

*Syntax:* <string>

*Description:* The name of the scripted search command to run, as defined in the `commands.conf` file.

**Optional arguments**

maxinputs

*Syntax:* maxinputs=<int>

*Description:* Specifies how many of the input results are passed to the script.

*Default:* 100

script-arg

*Syntax:* <string> ...

*Description:* One or more arguments to pass to the script. If you are passing multiple arguments, delimit each argument with a space.

**Usage**

The `script` command is effectively an alternative way to invoke custom search commands. See About writing custom search commands.

The following search

| script commandname

is largely synonymous with

| commandname

**Note:** Some functions of the script command have been removed over time. The explicit choice of Perl or Python as an argument is no longer functional and such an argument is ignored. If you need to write Perl search commands you need to declare them as Perl in the `commands.conf` file. This is not recommended, as you need to determine a number of undocumented things about the input and output formats. Additionally, support for explicit filename reference for scripts in the `etc/searchscripts` directory has been removed. All search commands must now be declared in the `commands.conf` file.
Examples

Example 1:

Run the Python script "myscript" with arguments, myarg1 and myarg2; then, email the results.

```
... | script myscript myarg1 myarg2 | sendemail to=david@splunk.com
```

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the script command.

scrub

Description

Anonymizes the search results by replacing identifying data - usernames, IP addresses, domain names, and so forth - with fictional values that maintain the same word length. For example, it might turn the string `user=carol@adalberto.com` into `user=aname@mycompany.com`. This lets Splunk users share log data without revealing confidential or personal information.

See the Usage section for more information.

Syntax

```
scrub [public-terms=<filename>] [private-terms=<filename>]
[name-terms=<filename>] [dictionary=<filename>] [timeconfig=<filename>]
[namespace=<string>]
```

Required arguments

None

Optional arguments

public-terms

**Syntax:** public-terms=<filename>
Description: Specify a filename that includes the public terms NOT to anonymize.

private-terms
Syntax: private-terms=<filename>
Description: Specify a filename that includes the private terms to anonymize.

name-terms
Syntax: name-terms=<filename>
Description: Specify a filename that includes the names to anonymize.

dictionary
Syntax: dictionary=<filename>
Description: Specify a filename that includes a dictionary of terms NOT to anonymize, unless those terms are in the private-terms file.

timeconfig
Syntax: timeconfig=<filename>
Description: Specify a filename that includes the time configurations to anonymize.

namespace
Syntax: namespace=<string>
Description: Specify an application that contains the alternative files to use for anonymizing, instead of using the built-in anonymizing files.

Usage

By default, the scrub command uses the dictionary and configuration files that are located in the $SPLUNK_HOME/etc/anonymizer directory. These default files can be overridden by specifying arguments to the scrub command. The arguments exactly correspond to the settings in the splunk anonymize CLI command. For details, issue the splunk help anonymize command.

You can add your own versions of the configuration files to the default location.

Alternatively, you can specify an application where you maintain your own copy of the dictionary and configuration files. To specify the application, use the namespace=<string> argument, where <string> is the name of the application that corresponds to the name that appears in the path $SPLUNK_HOME/etc/apps/<app>/anonymizer.
If the $SPLUNK_HOME/etc/apps/<app>/anonymizer directory does not exist, the Splunk software looks for the files in the $SPLUNK_HOME/etc/slave-apps/<app>/anonymizer directory.

The `scrub` command anonymizes all attributes, except those that start with underscore (_ ) except `_raw` or start with `date_`. Additionally, the following attributes are not anonymized: `eventtype`, `linecount`, `punct`, `sourcetype`, `timeendpos`, `timestartpos`.

The `scrub` command adheres to the default `maxresultrows` limit of 50000 results. This setting is documented in the `limits.conf` file in the `[searchresults]` stanza. See `limits.conf` in the Admin Manual.

Examples

1. Anonymize the current search results using the default files.
   
   ```
   ... | scrub
   ```

2. Anonymize the current search results using the specified private-terms file.
   
   This search uses the `abc_private-terms` file that is located in the `$SPLUNK_HOME/etc/anonymizer` directory.
   
   ```
   ... | scrub private-file=abc_private-terms
   ```

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `scrub` command.

search

Description

Use the `search` command to retrieve events from indexes or filter the results of a previous search command in the pipeline. You can retrieve events from your indexes, using keywords, quoted phrases, wildcards, and field-value expressions. The `search` command is implied at the beginning of any search. You do not need to specify the `search` command at the beginning of your search.
criteria.

When the `search` command is not the first command in the pipeline, the `search` command is used to filter the results of the previous command.

The search command can also be used in a subsearch. See about subsearches in the Search Manual.

After you retrieve events, you can apply commands to transform, filter, and report on the events. Use the vertical bar ( | ) , or pipe character, to apply a command to the retrieved events.

**Syntax**

`search <logical-expression>`

**Required arguments**

`<logical-expression>`  
**Syntax:** `<logical-expression> | <time-opts> | <search-modifier> | NOT <logical-expression> | <index-expression> | <comparison-expression> | <logical-expression> [OR] <logical-expression>  
**Description:** Includes all keywords or field-value pairs used to describe the events to retrieve from the index. Include parenthesis as necessary. Use Boolean expressions, comparison operators, time modifiers, search modifiers, or combinations of expressions for this argument.

**Logical expression options**

`<comparison-expression>`  
**Syntax:** `<field><comparison-operator><value> | <field> IN (<value-list>)  
**Description:** Compare a field to a literal value or provide a list of values that can appear in the field.

`<index-expression>`  
**Syntax:** "<string>" | <term> | <search-modifier>  
**Description:** Describe the events you want to retrieve from the index using literal strings and search modifiers.

`<time-opts>`  
**Syntax:** `[<timeformat>] (<time-modifier>)...  
**Description:** Describe the format of the starttime and endtime terms of the search. See Time options.
**Comparison expression options**

<comparison-operator>

**Syntax:** = | != | < | <= | > | >=  
**Description:** You can use comparison operators when searching field/value pairs. Comparison expressions with the equal ( = ) or not equal ( != ) operator compare string values. For example, "1" does not match "1.0". Comparison expressions with greater than or less than operators < > <= >= numerically compare two numbers and lexicographically compare other values. See Usage.

<field>

**Syntax:** <string>  
**Description:** The name of a field.

[value]

**Syntax:** <literal-value>  
**Description:** In comparison-expressions, the literal number or string value of a field.

[value-list]

**Syntax:** (<literal-value>, <literal-value>, ...)  
**Description:** Used with the IN operator to specify two or more values. For example use error IN (400, 402, 404, 406) instead of error=400 OR error=402 OR error=404 OR error=406

**Index expression options**

<string>

**Syntax:** "<string>"  
**Description:** Specify keywords or quoted phrases to match. When searching for strings and quoted strings (anything that's not a search modifier), Splunk software searches the _raw field for the matching events or results.

<search-modifier>

**Syntax:** <sourcetype-specifier> | <host-specifier> | <hosttag-specifier> | <source-specifier> | <savedsplunk-specifier> | <eventtype-specifier> | <eventtypetag-specifier> | <splunk_server-specifier>  
**Description:** Search for events from specified fields or field tags. For example, search for one or a combination of hosts, sources, source types, saved searches, and event types. Also, search for the field tag, with the format: tag::<field>=<string>.
◊ Read more about searching with default fields in the *Knowledge Manager manual*.
◊ Read more about using tags and field aliases in the *Knowledge Manager manual*.

```xml
<sourcetype-specifier>
  Syntax: sourcetype=<string>
  Description: Search for events from the specified sourcetype field.
</sourcetype-specifier>

<host-specifier>
  Syntax: host=<string>
  Description: Search for events from the specified host field.
</host-specifier>

<hosttag-specifier>
  Syntax: hosttag=<string>
  Description: Search for events that have hosts that are tagged by the string.
</hosttag-specifier>

<eventtype-specifier>
  Syntax: eventtype=<string>
  Description: Search for events that match the specified event type.
</eventtype-specifier>

<eventtypetag-specifier>
  Syntax: eventtypetag=<string>
  Description: Search for events that would match all eventtypes tagged by the string.
</eventtypetag-specifier>

<savedsplunk-specifier>
  Syntax: savedsearch=<string> | savedsplunk=<string>
  Description: Search for events that would be found by the specified saved search.
</savedsplunk-specifier>

<source-specifier>
  Syntax: source=<string>
  Description: Search for events from the specified source field.
</source-specifier>

<splunk_server-specifier>
  Syntax: splunk_server=<string>
  Description: Search for events from a specific server. Use "local" to refer to the search head.
</splunk_server-specifier>
```
### Time options

For a list of time modifiers, see [Time modifiers for search](#).

### <timeformat>

**Syntax:** `timeformat=<string>`

**Description:** Set the time format for starttime and endtime terms.

**Default:** `timeformat=%m/%d/%Y:%H:%M:%S`.

### <time-modifier>

**Syntax:**

- `starttime=<string>`
- `endtime=<string>`
- `earliest=<time_modifier>`
- `latest=<time_modifier>`

**Description:** Specify start and end times using relative or absolute time.

You can also use the earliest and latest attributes to specify absolute and relative time ranges for your search. For more about this time modifier syntax, see [Specify time modifiers in your search](#) in the Search Manual.

#### startime

**Syntax:** `starttime=<string>`

**Description:** Events must be later or equal to this time. Must match `timeformat`.

#### endtime

**Syntax:** `endtime=<string>`

**Description:** All events must be earlier or equal to this time.

### Usage

The `search` command is an **event-generating command** when it is the first command in the search, before the first pipe. When the `search` command is used further down the pipeline, it is a **distributable streaming command**. See [Command types](#).

When `search` is the first command in the search, you can use terms such as keywords, phrases, fields, boolean expressions, and comparison expressions to specify exactly which events you want to retrieve from Splunk indexes.

Some examples of search terms are:

- **keywords:** `error login`
- **quoted phrases:** "database error"
- **boolean operators:** `login NOT (error OR fail)`
To search field values that are SPL operators or keywords, such as `country=IN`, `country=AS`, `iso=AND`, or `state=OR`, you must enclose the operator or keyword in quotation marks. For example: `country="IN"`.

See Use the search command in the *Search Manual*.

**Boolean expressions**

The order in which Boolean expressions are evaluated with the `search` is:

1. Expressions within parentheses
2. NOT clauses
3. OR clauses
4. AND clauses

This evaluation order is different than the order used with the `where` command. The `where` command evaluates AND clauses before OR clauses.

**Comparing two fields**

To compare two fields, do not specify `index=myindex fieldA=fieldB` or `index=myindex fieldA!=fieldB` with the `search` command. When specifying a comparison expression, the `search` command expects a `<field>` compared with a `<value>`. The `search` command interprets `fieldB` as the value, and not as the name of a field.

Use the `where` command to compare two fields.

```
index=myindex | where fieldA=fieldB
```

For not equal comparisons, you can specify the criteria in several ways.

```
index=myindex | where fieldA!=fieldB
```

or

```
index=myindex | where NOT fieldA=fieldB
```
See Difference between NOT and != in the *Search Manual*.

**Multiple field-value comparisons with the IN operator**

Use the IN operator when you want to determine if a field contains one of several values.

For example, use this syntax:

```
... error_code IN (400, 402, 404, 406) | ...
```

Instead of this syntax:

```
... error_code=400 OR error_code=402 OR error_code=404 OR
error_code=406 | ...
```

When used with the *search* command, you can use a wildcard character in the list of values for the IN operator. For example:

```
... error_code IN (40*) | ...
```

You can use the NOT operator with the IN operator. For example:

```
... NOT clientip IN (211.166.11.101, 182.236.164.11,
128.241.220.82) | ...
```

There is also an IN function that you can use with the *eval* and *where* commands. Wild card characters are not allowed in the values list when the IN function is used with the *eval* and *where* commands. See *Comparison and Conditional functions*.

**Lexicographical order**

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.

- Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically as 10, 100, 70, 9.
- Uppercase letters are sorted before lowercase letters.
- Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

You can specify a custom sort order that overrides the lexicographical order. See the blog *Order Up! Custom Sort Orders*. 548
**Quotes and escaping characters**

In general, you need quotation marks around phrases and field values that include white spaces, commas, pipes, quotations, and brackets. Quotation marks must be balanced. An opening quotation must be followed by an unescaped closing quotation. For example:

- A search such as `error | stats count` will find the number of events containing the string `error`.
- A search such as `... | search "error | stats count"` would return the raw events containing `error`, a pipe, `stats`, and `count`, in that order.

Additionally, you want to use quotation marks around keywords and phrases if you do not want to search for their default meaning, such as Boolean operators and field/value pairs. For example:

- A search for the keyword AND without meaning the Boolean operator: `error "AND"`
- A search for this field/value phrase: `error "startswith=foo"`

The backslash character (`\`) is used to escape quotes, pipes, and itself. Backslash escape sequences are still expanded inside quotation marks. For example:

- The sequence `\|` as part of a search will send a pipe character to the command, instead of having the pipe split between commands.
- The sequence `\"` will send a literal quotation mark to the command, for example for searching for a literal quotation mark or inserting a literal quotation mark into a field using rex.
- The `\\` sequence will be available as a literal backslash in the command.

Unrecognized backslash sequences are not altered:

- For example `\s` in a search string will be available as `\s` to the command, because `\s` is not a known escape sequence.
- However, in the search string `\\s` will be available as `\s` to the command, because `\\` is a known escape sequence that is converted to `\`.
**Search with TERM()**

You can use the TERM() directive to force Splunk software to match whatever is inside the parentheses as a single term in the index. TERM is more useful when the term contains minor segmenters, such as periods, and is bounded by major segmenters, such as spaces or commas. In fact, TERM does not work for terms that are not bounded by major breakers.

See Use CASE and TERM to match phrases in the *Search Manual*.

**Search with CASE()**

You can use the CASE() directive to search for terms and field values that are case-sensitive.

See Use CASE and TERM to match phrases in the *Search Manual*.

**Examples**

These examples demonstrate how to use the search command. You can find more examples in the Start Searching topic of the Search Tutorial.

1. **Field-value pair matching**

This example demonstrates field-value pair matching for specific values of source IP (src) and destination IP (dst).

```
src="10.9.165.*" OR dst="10.9.165.8"
```

2. **Using boolean and comparison operators**

This example demonstrates field-value pair matching with boolean and comparison operators. Search for events with code values of either 10 or 29, and any host that isn’t "localhost", and an xqp value that is greater than 5.

```
(code=10 OR code=29) host!="localhost" xqp>5
```

In this example you could also use the IN operator since you are specifying two field-value pairs on the same field. The revised search is:

```
code IN(10, 29) host!="localhost" xqp>5
```
3. Using wildcards

This example demonstrates field-value pair matching with wildcards. Search for events from all the web servers that have an HTTP client or server error status.

host=webserver* (status=4* OR status=5*)

In this example you could also use the IN operator since you are specifying two field-value pairs on the same field. The revised search is:

host=webserver* status IN(4*, 5*)

4. Using the IN operator

This example shows how to use the IN operator to specify a list of field-value pair matchings. In the events from an access.log file, search the action field for the values addtocart or purchase.

sourcetype=access_combined_wcookie action IN (addtocart, purchase)

5. Specifying a secondary search

This example uses the search command twice. The search command is implied at the beginning of every search with the criteria eventtype=web-traffic. The search command is used again later in the search pipeline to filter out the results. This search defines a web session using the transaction command and searches for the user sessions that contain more than three events.

eventtype=web-traffic | transaction clientip startswith="login" endswith="logout" | search eventcount>3

6. Using the NOT or != comparisons

Searching with the boolean "NOT" comparison operator is not the same as using the "!=" comparison.

The following search returns everything except fieldA="value2", including all other fields.

NOT fieldA="value2"

The following search returns events where fieldA exists and does not have the value "value2".
fieldA!="value2"

If you use a wildcard for the value, NOT fieldA=* returns events where fieldA is null or undefined, and fieldA!=* never returns any events.

See Difference between NOT and != in the Search Manual.

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the search command.

searchtxn

Description

Efficiently returns transaction events that match a transaction type and contain specific text. If you have Splunk Cloud and want to define transaction types, file a Support ticket.

Syntax

| searchtxn <transaction-name> [max_terms=<int>] [use_disjunct=<bool>] [eventsonly=<bool>] <search-string>

Required arguments

<transaction-name>
  Syntax: <transactiontype>
  Description: The name of the transaction type stanza that is defined in transactiontypes.conf.

<search-string>
  Syntax: <string>
  Description: Terms to search for within the transaction events.

Optional arguments

eventsonly
  Syntax: eventsonly=<bool>
**Description:** If true, retrieves only the relevant events but does not run "| transaction" command.
**Default:** false

**max_terms**

**Syntax:** maxterms=<int>

**Description:** Integer between 1-1000 which determines how many unique field values all fields can use. Using smaller values speeds up search, favoring more recent values.

**Default:** 1000

**use_disjunct**

**Syntax:** use_disjunct=<bool>

**Description:** Specifies if each term in <search-string> should be processed as if separated by an OR operator on the initial search.

**Default:** true

**Usage**

The `searchtxn` command is an **event-generating command**. See Command types.

Generating commands use a leading pipe character and should be the first command in a search.

**Transactions**

The command works only for transactions bound together by particular field values, not by ordering or time constraints.

Suppose you have a `<transactiontype>` stanza in the `transactiontypes.conf.in` file called "email". The stanza contains the following settings.

- `fields=qid, pid`
- `search=sourcetype=sendmail_syslog to=root`

The `searchtxn` command finds all of the events that match `sourcetype="sendmail_syslog" to=root`.

From those results, all fields that contain a qid or pid located are used to further search for relevant transaction events. When no additional qid or pid values are found, the resulting search is run:
Examples

Example 1:

Find all email transactions to root from David Smith.

| searchtxn email to=root from="David Smith"

See also

transaction

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the searchtxn command.

delfjoin

Description

Join search result rows with other search result rows in the same result set, based on one or more fields that you specify.

Syntax

selfjoin [<selfjoin-options>...] <field-list>

Required arguments

<field-list>

Syntax: <field>...

Description: The field or list of fields to join on.

Optional arguments

<selfjoin-options>

Syntax: overwrite=<bool> | max=<int> | keepsingle=<bool>
**Description:** Options that control the search result set that is returned. You can specify one or more of these options.

**Selfjoin options**

**keepsingle**

**Syntax:** keepsingle=<bool>

**Description:** Controls whether or not to retain results that have with a unique value in the join fields. When `keepsingle=true` search results that have no other results to join with are kept in the output.

**Default:** false

**max**

**Syntax:** max=<int>

**Description:** Indicates the maximum number of 'other' results to join with each main result. If `max=0`, there is no limit. This argument sets the maximum for the 'other' results. The maximum number of main results is 100,000.

**Default:** 1

**overwrite**

**Syntax:** overwrite=<bool>

**Description:** When `overwrite=true`, causes fields from the 'other' results to overwrite fields of the main results. The main results are used as the basis for the join.

**Default:** true

**Usage**

Self joins are more commonly used with relational database tables. They are used less commonly with event data.

An example of an events usecase is with events that contain information about processes, where each process has a parent process ID. You can use the `selfjoin` command to correlate information about a process with information about the parent process.

See the [Extended example](#).

**Basic example**
**1: Use a single field to join results**

Join the results with itself on the 'id' field.

```
... | selfjoin id
```

**Extended example**

The following example shows how the `selfjoin` command works against a simple set of results. You can follow along with this example on your own Splunk instance.

This example builds a search incrementally. With each addition to the search, the search is rerun and the impact of the additions are shown in a results table. The values in the `_time` field change each time you rerun the search. However, in this example the values in the results table are not changed so that we can focus on how the changes to the search impact the results.

1. Start by creating a simple set of 5 results by using the `makeresults` command.

    ```
    | makeresults count=5
    ```

    There are 5 results created, each with the same timestamp.

    | _time     |
    |-----------|
    | 2018-01-18 14:38:59 |
    | 2018-01-18 14:38:59 |
    | 2018-01-18 14:38:59 |
    | 2018-01-18 14:38:59 |
    | 2018-01-18 14:38:59 |

2. To keep better track of each result use the `streamstats` command to add a field that numbers each result.

    ```
    | makeresults count=5 | streamstats count as a
    ```

    The `a` field is added to the results.

    | _time   | a     |
    |---------|-------|
    | 2018-01-18 14:38:59 | 1     |
    | 2018-01-18 14:38:59 | 2     |
3. Additionally, use the `eval` command to change the timestamps to be 60 seconds apart. Different timestamps makes this example more realistic.

```
| makeresults count=5 | streamstats count as a | eval _time = _time + (60*a)
```

The minute portion of the timestamp is updated.

```
<table>
<thead>
<tr>
<th>_time</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-01-18 14:38:59</td>
<td>3</td>
</tr>
<tr>
<td>2018-01-18 14:38:59</td>
<td>4</td>
</tr>
<tr>
<td>2018-01-18 14:38:59</td>
<td>5</td>
</tr>
</tbody>
</table>
```

4. Next use the `eval` command to create a field to use as the field to join the results on.

```
| makeresults count=5 | streamstats count as a | eval _time = _time + (60*a) | eval joiner="x"
```

The new field is added.

```
<table>
<thead>
<tr>
<th>_time</th>
<th>a</th>
<th>joiner</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-01-18 14:38:59</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:39:59</td>
<td>2</td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:40:59</td>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:41:59</td>
<td>4</td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:42:59</td>
<td>5</td>
<td>x</td>
</tr>
</tbody>
</table>
```

5. Use the `eval` command to create some fields with data.

An `if` function is used with a modulo (modulus) operation to add different data to each of the new fields. A modulo operation finds the remainder after the division of one number by another number:
• The `eval b` command processes each result and performs a modulo operation. If the remainder of \( a/2 \) is 0, put "something" into the field "b", otherwise put "nada" into field "b".

• The `eval c` command processes each result and performs a modulo operation. If the remainder \( a/2 \) is 1, put "something else" into the field "c", otherwise put nothing (NULL) into field "c".

```
| makeresults count=5 | streamstats count as a | eval _time = _time + (60*a) | eval joiner="x" | eval b = if(a%2==0,"something","nada"), c = if(a%2==1,"somethingelse",null())
```

The new fields are added and the fields are arranged in alphabetical order by field name, except for the `_time` field.

| _time          | a   | b                  | c                  | joiner
|----------------|-----|--------------------|--------------------|-------
| 2018-01-18 14:38:59 | 1   | nada               | somethingelse      | x
| 2018-01-18 14:39:59 | 2   | something          |                    | x
| 2018-01-18 14:40:59 | 3   | nada               | somethingelse      | x
| 2018-01-18 14:41:59 | 4   | something          |                    | x
| 2018-01-18 14:42:59 | 5   | nada               | somethingelse      | x

6. Use the `selfjoin` command to join the results on the `joiner` field.

```
| makeresults count=5 | streamstats count as a | eval _time = _time + (60*a) | eval joiner="x" | eval b = if(a%2==0,"something","nada"), c = if(a%2==1,"somethingelse",null()) | selfjoin joiner
```

The results are joined.

| _time          | a   | b                  | c                  | joiner
|----------------|-----|--------------------|--------------------|-------
| 2018-01-18 14:39:59 | 2   | something          | somethingelse      | x
| 2018-01-18 14:40:59 | 3   | nada               | somethingelse      | x
| 2018-01-18 14:41:59 | 4   | something          | somethingelse      | x
| 2018-01-18 14:42:59 | 5   | nada               | somethingelse      | x

7. To understand how the `selfjoin` command joins the results together, remove the `| selfjoin joiner` portion of the search. Then modify the search to append the values from the `a` field to the values in the `b` and `c` fields.

```
| makeresults count=5 | streamstats count as a | eval _time = _time + (60*a) | eval joiner="x" | eval b = if(a%2==0,"something"+a,"nada"+a), c = if(a%2==1,"somethingelse"+a,null())
```

558
The results now have the row number appended to the values in the \( \texttt{b} \) and \( \texttt{c} \) fields.

<table>
<thead>
<tr>
<th><code>_time</code></th>
<th>( \texttt{a} )</th>
<th>( \texttt{b} )</th>
<th>( \texttt{c} )</th>
<th>( \texttt{joiner} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-01-18 14:38:59</td>
<td>1</td>
<td>nada1</td>
<td>somethingelse1</td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:39:59</td>
<td>2</td>
<td>something2</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:40:59</td>
<td>3</td>
<td>nada3</td>
<td>somethingelse3</td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:41:59</td>
<td>4</td>
<td>something4</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:42:59</td>
<td>5</td>
<td>nada5</td>
<td>somethingelse5</td>
<td>x</td>
</tr>
</tbody>
</table>

8. Now add the \texttt{selfjoin} command back into the search.

```bash
<table>
<thead>
<tr>
<th>makeresults count=5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

The results of the self join.

<table>
<thead>
<tr>
<th><code>_time</code></th>
<th>( \texttt{a} )</th>
<th>( \texttt{b} )</th>
<th>( \texttt{c} )</th>
<th>( \texttt{joiner} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-01-18 14:39:59</td>
<td>2</td>
<td>something2</td>
<td>somethingelse1</td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:40:59</td>
<td>3</td>
<td>nada3</td>
<td>somethingelse3</td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:41:59</td>
<td>4</td>
<td>something4</td>
<td>somethingelse3</td>
<td>x</td>
</tr>
<tr>
<td>2018-01-18 14:42:59</td>
<td>5</td>
<td>nada5</td>
<td>somethingelse5</td>
<td>x</td>
</tr>
</tbody>
</table>

If there are values for a field in both rows, the last result row, based on the `_time` value, takes precedence. The joins performed are shown in the following table.

<table>
<thead>
<tr>
<th>Result row</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Row 1 is joined with row 2 and returned as row 2.</td>
<td>In field ( \texttt{b} ), the value \texttt{nada1} is discarded because the value \texttt{something2} in row 2 takes precedence. In field ( \texttt{c} ), there is no value in row 2. The value \texttt{somethingelse1} from row 1 is returned.</td>
</tr>
<tr>
<td>2</td>
<td>Row 2 is joined with row 3 and returned as row 3.</td>
<td>Since row 3 contains values for both field ( \texttt{b} ) and field ( \texttt{c} ), the values in row 3 take precedence and the values in row 2 are discarded.</td>
</tr>
<tr>
<td>3</td>
<td>Row 3 is joined with row 4 and</td>
<td>In field ( \texttt{b} ), the value \texttt{nada3} is discarded because the value \texttt{something4} in row 4 takes precedence. In</td>
</tr>
<tr>
<td>Result row</td>
<td>Output</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>4</td>
<td>returned as row 4.</td>
<td>field c, there is no value in row 4. The value somethingelse3 from row 3 is returned.</td>
</tr>
<tr>
<td>5</td>
<td>Row 4 is joined with row 5 and returned as row 5.</td>
<td>Since row 5 contains values for both field b and field c, the values in row 5 take precedence and the values in row 4 are discarded.</td>
</tr>
<tr>
<td>5</td>
<td>Row 5 has no other row to join with.</td>
<td>No additional results are returned.</td>
</tr>
</tbody>
</table>

(Thanks to Splunk user Alacercogitatus for helping with this example.)

**See also**

join

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the selfjoin command.

**sendemail**

**Description**

Use the sendemail command to generate email notifications. You can email search results to specified email addresses.

**Syntax**

sendemail to=<email_list>

[from=<email_list>]
[cc=<email_list>]
[bcc=<email_list>]
[subject=<string>]
[format=csv | table | raw]
[inline= <bool>]

560
[sendresults=<bool>]
[sendpdf=<bool>]
[priority=highest | high | normal | low | lowest]
[server=<string>]
[width_sort_columns=<bool>]
[graceful=<bool>]
[content_type=html | plain]
[message=<string>]
[sendcsv=<bool>]
[use_ssl=<bool>]
[use_tls=<bool>]
[pdfview=<string>]
[papersize=letter | legal | ledger | a2 | a3 | a4 | a5]
[paperorientation=portrait | landscape]
[maxinputs=<int>]
[maxtime=<int> m | s | h | d]
[footer=<string>]

**Required arguments**

to

**Syntax:** to=<email_list>

**Description:** List of email addresses to send search results to. Specify email addresses in a comma-separated and quoted list. For example: "alex@email.com, maria@email.com, wei@email.com"

**Optional arguments**

bcc

**Syntax:** bcc=<email_list>

**Description:** Blind courtesy copy line. Specify email addresses in a comma-separated and quoted list.

cc

**Syntax:** cc=<email_list>

**Description:** Courtesy copy line. Specify email addresses in a comma-separated and quoted list.

content_type

**Syntax:** content_type=html | plain

**Description:** The format type of the email.

**Default:** The default value for the `content_type` argument is set in the [email] stanza of the `alert_actions.conf` file. The default value for a new
or upgraded Splunk installation is html.

**format**

**Syntax:** format=csv | raw | table  
**Description:** Specifies how to format inline results.  
**Default:** The default value for the format argument is set in the [email] stanza of the alert_actions.conf file. The default value for a new or upgraded Splunk installation is table.

**footer**

**Syntax:** footer=<string>  
**Description:** Specify an alternate email footer.  
**Default:** The default footer is:  
If you believe you've received this email in error, please see your Splunk administrator. splunk > the engine for machine data.

To force a new line in the footer, use Shift+Enter.

**from**

**Syntax:** from=<email_list>  
**Description:** Email address from line.  
**Default:** "splunk@<hostname>"

**inline**

**Syntax:** inline=<boolean>  
**Description:** Specifies whether to send the results in the message body or as an attachment. By default, an attachment is provided as a CSV file. See the Usage section.  
**Default:** The default value for the inline argument is set in the [email] stanza of the alert_actions.conf file. The default value for a new or upgraded Splunk installation is false.

**graceful**

**Syntax:** graceful=<boolean>  
**Description:** If set to true, no error is returned if sending the email fails for whatever reason. The remainder of the search continues as if the the sendemail command was not part of the search. If graceful=false and sending the email fails, the search returns an error.  
**Default:** false

**maxinputs**

**Syntax:** maxinputs=<integer>
Description: Set the maximum number of search results sent via alerts.
Default: 50000

maxtime
Syntax: maxtime=<integer>m | s | h | d
Description: The maximum amount of time that the execution of an action is allowed to take before the action is aborted.
Example: 2m
Default: no limit

message
Syntax: message=<string>
Description: Specifies the message sent in the email.
Default: The default message depends on which other arguments are specified with the sendemail command.
◊ If sendresults=false the message defaults to "Search complete."
◊ If sendresults=true, inline=true, and either sendpdf=false or sendcsv=false, message defaults to "Search results."
◊ If sendpdf=true or sendcsv=true, message defaults to "Search results attached."

paperorientation
Syntax: paperorientation=portrait | landscape
Description: The orientation of the paper.
Default: portrait

papersize
Syntax: papersize=letter | legal | ledger | a2 | a3 | a4 | a5
Description: Default paper size for PDFs. Acceptable values: letter, legal, ledger, a2, a3, a4, a5.
Default: letter

pdfview
Syntax: pdfview=<string>
Description: Name of a view.xml file to send as a PDF. For example, mydashboard.xml, search.xml, or foo.xml. Generally this is the name of a dashboard, but it could also be the name of a single page application or some other object. Specify the name only. Do not specify the filename extension. The view.xml files are located in <SPLUNK_HOME>/data/ui/views.

priority
Syntax: priority=highest | high | normal | low | lowest
**Description**: Set the priority of the email as it appears in the email client. Lowest or 5, low or 4, high or 2, highest or 1.

**Default**: normal or 3

**sendcsv**

**Syntax**: sendcsv=<boolean>

**Description**: Specify whether to send the results with the email as an attached CSV file or not.

**Default**: The default value for the sendcsv argument is set in the [email] stanza of the alert_actions.conf file. The default value for a new or upgraded Splunk installation is false.

**sendpdf**

**Syntax**: sendpdf=<boolean>

**Description**: Specify whether to send the results with the email as an attached PDF or not. For more information about generating PDFs, see "Generate PDFs of your reports and dashboards" in the Reporting Manual.

**Default**: The default value for the sendpdf argument is set in the [email] stanza of the alert_actions.conf file. The default value for a new or upgraded Splunk installation is false.

**sendresults**

**Syntax**: sendresults=<boolean>

**Description**: Determines whether the results should be included with the email. See the Usage section.

**Default**: The default value for the sendresults argument is set in the [email] stanza of the alert_actions.conf file. The default value for a new or upgraded Splunk installation is false.

**server**

**Syntax**: server=<string>

**Description**: If the SMTP server is not local, use this to specify it.

**Default**: localhost

**subject**

**Syntax**: subject=<string>

**Description**: Specifies the subject line.

**Default**: "Splunk Results"

**use_ssl**

**Syntax**: use_ssl=<boolean>
**Description:** Whether to use SSL when communicating with the SMTP server. When set to 1 (true), you must also specify both the server name or IP address and the TCP port in the "mailserver" attribute.

**Default:** false

*use_tls*

**Syntax:** use_tls=<boolean>

**Description:** Specify whether to use TLS (transport layer security) when communicating with the SMTP server (starttls).

**Default:** false

*width_sort_columns*

**Syntax:** width_sort_columns=<boolean>

**Description:** This is only valid for plain text emails. Specifies whether the columns should be sorted by their width.

**Default:** true

**Usage**

If you set sendresults=true and inline=false and do not specify format, a CSV file is attached to the email.

**Examples**

1: **Send search results to the specified email**

Send search results to the specified email. By default, the results are formatted as a table.

```plaintext
... | sendemail to="elvis@splunk.com" sendresults=true
```

2: **Send search results in table format**

Send search results in a raw format with the subject "myresults".

```plaintext
... | sendemail to="elvis@splunk.com, john@splunk.com" format=raw subject=myresults server=mail.splunk.com sendresults=true
```

3. **Include a PDF attachment, a message, and raw inline results**

Send an email notification with a PDF attachment, a message, and raw inline results.
index=_internal | head 5 | sendemail to=example@splunk.com server@mail.example.com subject="Here is an email from Splunk" message="This is an example message" sendresults=true inline=true format=raw sendpdf=true

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the sendemail command.

set

Description

Performs set operations on subsearches.

Syntax

| set (union | diff | intersect) subsearch subsearch

Required arguments

union | diff | intersect

Syntax: union | diff | intersect
Description: Performs two subsearches, then executes the specified set operation on the two sets of search results.
* Specify union to return results from either subsearch.
* Specify diff to return results from only one of the two subsearches.
There is no information provided as to which subsearch the results originated from.
* Specify intersect to return results that are common to both subsearches.

subsearch

Syntax: ":[ [ <string> ] ]"
Description: Specifies a subsearch. Subsearches must be enclosed in square brackets. For more information about subsearch syntax, see "About subsearches" in the Search Manual.
Usage

The set command is an event-generating command. See Command types.

Generating commands use a leading pipe character and should be the first command in a search.

Results

The set command considers results to be the same if all of fields that the results contain match. Some internal fields generated by the search, such as _serial, vary from search to search. You need to filter out some of the fields if you are using the set command with raw events, as opposed to transformed results such as those from a stats command. Typically in these cases, all fields are the same from search to search.

Output limitations

There is a limit on the quantity of results that come out of the invoked subsearches that the set command receives to operate on. If this limit is exceeded, the input result set to the diff command is silently truncated.

If you have Splunk Enterprise, you can adjust this limit by editing the limits.conf file and changing the maxout value in the [subsearch] stanza. If this value is altered, the default quantity of results coming from a variety of subsearch scenarios are altered. Note that very large values might cause extensive stalls during the 'parsing' phase of a search, which is when subsearches run. The default value for this limit is 10000.

Only users with file system access, such as system administrators, can edit the configuration files. Never change or copy the configuration files in the default directory. The files in the default directory must remain intact and in their original location. Make the changes in the local directory.

See How to edit a configuration file.

If you are using Splunk Cloud and want to edit a configuration file, file a Support ticket.

Result rows limitations

By default the set command attempts to traverse a maximum of 50000 items from each subsearch. If the number of input results from either search exceeds
this limit, the set command silently ignores the remaining events. By default, the maxout setting for subsearches prevents the number of results from exceeding this limit.

This maximum is controlled by the maxresultrows setting in the [set] stanza in the limits.conf file. Increasing this limit can result in more memory usage.

Only users with file system access, such as system administrators, can edit the configuration files. Never change or copy the configuration files in the default directory. The files in the default directory must remain intact and in their original location. Make the changes in the local directory.

See How to edit a configuration file.

If you are using Splunk Cloud and want to edit a configuration file, file a Support ticket.

**Examples**

**Example 1:**

Return values of "URL" that contain the string "404" or "303" but not both.

| set diff [search 404 | fields url] [search 303 | fields url]

**Example 2:**

Return all urls that have 404 errors and 303 errors.

| set intersect [search 404 | fields url] [search 303 | fields url]

**Note:** When you use the fields command in your subsearches, it does not filter out internal fields by default. If you do not want the set command to compare internal fields, such as the _raw or _time fields, you need to explicitly exclude them from the subsearches:

| set intersect [search 404 | fields url | fields - _*] [search 303 | fields url | fields - _*]

**See also**

append, appendcols, appendpipe, join, diff
**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the set command.

**setfields**

**Description**

Sets the field values for all results to a common value.

Sets the value of the given fields to the specified values for each event in the result set. Delimit multiple definitions with commas. Missing fields are added, present fields are overwritten.

Whenever you need to change or define field values, you can use the more general purpose `eval` command. See usage of an `eval expression` to set the value of a field in Example 1.

**Syntax**

`setfields <setfields-arg>, ...`

**Required arguments**

`<setfields-arg>`

- **Syntax**: `string="<string>", ...`
- **Description**: A key-value pair, with the value quoted. If you specify multiple key-value pairs, separate each pair with a comma. Standard key cleaning is performed. This means all non-alphanumeric characters are replaced with '_' and leading '_' are removed.

**Examples**

**Example 1:**

Specify a value for the ip and foo fields.

```shell
... | setfields ip="10.10.10.10", foo="foo bar"
```

To do this with the `eval` command:
... | eval ip="10.10.10.10" | eval foo="foo bar"

See also
eval, fillnull, rename

Answers
Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the setfields command.

sichart

**Summary indexing** is a method you can use to speed up long-running searches that do not qualify for report acceleration, such as searches that use commands that are not **streamable** before the reporting command. For more information, see "About report acceleration and summary indexing" and "Use summary indexing for increased reporting efficiency" in the *Knowledge Manager Manual*.

**Description**

The summary indexing version of the **chart** command. The **sichart** command populates a summary index with the statistics necessary to generate a chart visualization. For example, it can create a column, line, area, or pie chart. After you populate the summary index, you can use the **chart** command with the exact same search that you used with the **sichart** command to search against the summary index.

**Syntax**

```bash
sichart [sep=<string>] [format=<string>] [cont=<bool>] [limit=<int>]
[agg=<stats-agg-term>] ( <stats-agg-term> | <sparkline-agg-term> |
"("<eval-expression>")")... [ BY <field> [ <bins-options> ] [ <split-by-clause> ] ] | [ OVER <field> [ <bins-options> ] [BY <split-by-clause> ] ]
```

For syntax descriptions, refer to the **chart command**.

For information about functions that you can use with the **sichart** command, see Statistical and charting functions.
Examples

Example 1:

Compute the necessary information to later do 'chart avg(foo) by bar' on summary indexed results.

... | sichart avg(foo) by bar

See also

chart, collect, overlap, sirare, sistats, sitimechart, sitop

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the sichart command.

sirare

Summary indexing is a method you can use to speed up long-running searches that do not qualify for report acceleration, such as searches that use commands that are not streamable before the reporting command. For more information, see "About report acceleration and summary indexing" and "Use summary indexing for increased reporting efficiency" in the Knowledge Manager Manual.

Description

The sirare command is the summary indexing version of the rare command, which returns the least common values of a field or combination of fields. The sirare command populates a summary index with the statistics necessary to generate a rare report. After you populate the summary index, use the regular rare command with the exact same search string as the rare command search to report against it.

Syntax

sirare [<top-options>...] <field-list> [by-clause]
Required arguments

<string>,...

Description: Comma-delimited list of field names.

Optional arguments

BY <field-list>

Description: The name of one or more fields to group by.

<top-options>

countfield=<string> | limit=<int> | percentfield=<string> | showcount=<bool> | showperc=<bool>

Description: Options that specify the type and number of values to display. These are the same <top-options> used by the rare and top commands.

Top options

countfield

Syntax: countfield=<string>

Description: Name of a new field to write the value of count.

Default: "count"

limit

Syntax: limit=<int>

Description: Specifies how many tuples to return, "0" returns all values.

percentfield

Syntax: percentfield=<string>

Description: Name of a new field to write the value of percentage.

Default: "percent"

showcount

Syntax: showcount=<bool>

Description: Specify whether to create a field called "count" (see "countfield" option) with the count of that tuple.

Default: true

showpercent

Syntax: showpercent=<bool>
**Description:** Specify whether to create a field called "percent" (see "percentfield" option) with the relative prevalence of that tuple.

**Default:** true

**Examples**

**Example 1:**

Compute the necessary information to later do 'rare foo bar' on summary indexed results.

```
... | sirare foo bar
```

**See also**

collect, overlap, sichart, sistats, sitimechart, sitop

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the sirare command.

**sistats**

**Description**

The `sistats` command is one of several commands that you can use to create summary indexes. Summary indexing is one of the methods that you can use to speed up searches that take a long time to run.

The `sistats` command is the summary indexing version of the `stats` command, which calculates aggregate statistics over the dataset.

The `sistats` command populates a summary index. You must then create a report to generate the summary statistics. See the Usage section.

**Syntax**

```
sistats [allnum=<bool>] [delim=<string>] ( <stats-agg-term> | <sparkline-agg-term> ) [by clause>
```

573
• For descriptions of each of the arguments in this syntax, refer to the stats command.
• For information about functions that you can use with the sistats command, see Statistical and charting functions.

**Usage**

The summary indexes exist separately from your main indexes.

After you create the summary index, create a report by running a search against the summary index. You use the exact same search string that you used to populate the summary index, substituting the stats command for the sistats command, to create your reports.

For more information, see About report acceleration and summary indexing and Use summary indexing for increased reporting efficiency in the *Knowledge Manager Manual*.

**Memory and maximum results**

In the limits.conf file, the maxresultrows setting in the [searchresults] stanza specifies the maximum number of results to return. The default value is 50,000. Increasing this limit can result in more memory usage.

The max_mem_usage_mb setting in the [default] stanza is used to limit how much memory the sistats command uses to keep track of information. If the sistats command reaches this limit, the command stops adding the requested fields to the search results. You can increase the limit, contingent on the available system memory.

If you are using Splunk Cloud and want to change either of these limits, file a Support ticket.

**Examples**

**Example 1:**

Create a summary index with the statistics about the average, for each hour, of any unique field that ends with the string "lay". For example, delay, xdelay, relay, etc.

```
... | sistats avg(*lay) BY date_hour
```
To create a report, run a search against the summary index using this search

```plaintext
index=summary | stats avg(*lay) BY date_hour
```

**See also**

collect, overlap, sichart, sirare, sitop, sitimechart

For a detailed explanation and examples of summary indexing, see Use summary indexing for increased reporting efficiency in the Knowledge Manager Manual.

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the sistats command.

**sitimechart**

**Summary indexing** is a method you can use to speed up long-running searches that do not qualify for report acceleration, such as searches that use commands that are not streamable before the reporting command. For more information, see "About report acceleration and summary indexing" and "Use summary indexing for increased reporting efficiency" in the Knowledge Manager Manual.

**Description**

The sitimechart command is the summary indexing version of the timechart command, which creates a time-series chart visualization with the corresponding table of statistics. The sitimechart command populates a summary index with the statistics necessary to generate a timechart report. After you populate the summary index, use the regular timechart command with the exact same search string as the sitimechart command search to report against it.

**Syntax**

```plaintext
sitimechart [sep=<string>] [partial=<bool>] [cont=<bool>] [limit=<int>] [agg=<stats-agg-term>] [bin-options...] (<single-agg> [BY <split-by-clause>] ) | (<eval-expression> BY <split-by-clause>)
```
For syntax descriptions, refer to the `timechart` command.

For information about functions that you can use with the `timechart` command, see Statistical and charting functions.

**Examples**

**Example 1:**

Compute the necessary information to later do 'timechart avg(foo) by bar' on summary indexed results.

```plaintext
... | sitimechart avg(foo) by bar
```

**See also**

`collect`, `overlap`, `sichart`, `sirare`, `sistats`, `sitop`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `sitimechart` command.

**sitop**

**Summary indexing** is a method you can use to speed up long-running searches that do not qualify for report acceleration, such as searches that use commands that are not **streamable** before the reporting command. For more information, see "About report acceleration and summary indexing" and "Use summary indexing for increased reporting efficiency" in the *Knowledge Manager Manual*.

**Description**

The `sitop` command is the summary indexing version of the `top` command, which returns the most frequent value of a field or combination of fields. The `sitop` command populates a summary index with the statistics necessary to generate a top report. After you populate the summary index, use the regular `top` command with the exact same search string as the `sitop` command search to report against it.
Syntax

sitop [<N>] [top-options...] <field-list> [by-clause]

Note: This is the exact same syntax as that of the top command.

Required arguments

<field-list>
   Syntax: <field>, ...
   Description: Comma-delimited list of field names.

Optional arguments

<N>
   Syntax: <int>
   Description: The number of results to return.

<top-options>
   Syntax: countfield=<string> | limit=<int> | otherstr=<string> | percentfield=<string> | showcount=<bool> | showperc=<bool> | useother=<bool>
   Description: Options for the sitop command. See Top options.

<by-clause>
   Syntax: BY <field-list>
   Description: The name of one or more fields to group by.

Top options

countfield
   Syntax: countfield=<string>
   Description: The name of a new field that the value of count is written to.
   Default: count

limit
   Syntax: limit=<int>
   Description: Specifies how many tuples to return, "0" returns all values.
   Default: "10"

otherstr
   Syntax: otherstr=<string>
Description: If useother is true, specify the value that is written into the row representing all other values.
Default: "OTHER"

percentfield
Syntax: percentfield=<string>
Description: Name of a new field to write the value of percentage.
Default: "percent"

showcount
Syntax: showcount=<bool>
Description: Specify whether to create a field called "count" (see "countfield" option) with the count of that tuple.
Default: true

showperc
Syntax: showperc=<bool>
Description: Specify whether to create a field called "percent" (see "percentfield" option) with the relative prevalence of that tuple.
Default: true

useother
Syntax: useother=<bool>
Description: Specify whether or not to add a row that represents all values not included due to the limit cutoff.
Default: false

Examples

Example 1:
Compute the necessary information to later do 'top foo bar' on summary indexed results.

... | sitop foo bar

Example 2:
Populate a summary index with the top source IP addresses in a scheduled search that runs daily:

eventtype=firewall | sitop src_ip

578
Save the search as, "Summary - firewall top src_ip".

Later, when you want to retrieve that information and report on it, run this search over the past year:

```
index=summary search_name="summary - firewall top src_ip" | top src_ip
```

Additionally, because this search specifies the search name, it filters out other data that have been placed in the summary index by other summary indexing searches.

**See also**

`collect, overlap, sichart, sirare, sistats, sitimechart`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the sitop command.

**snowincident**

The `snowincident` command is used with the Splunk Add-on for ServiceNow.

For information about this command, see Use custom generating search commands for the Splunk Add-on for ServiceNow in *Splunk Add-on for ServiceNow*.

**snowincidentstream**

The `snowincidentstream` command is used with the Splunk Add-on for ServiceNow.

For information about this command, see Use custom streaming search commands for the Splunk Add-on for ServiceNow in *Splunk Add-on for ServiceNow*. 

579
snowevent

The `snowevent` command is used with the Splunk Add-on for ServiceNow.

For information about this command, see Use custom generating search commands for the Splunk Add-on for ServiceNow in Splunk Add-on for ServiceNow.

snoweventstream

The `snoweventstream` command is used with the Splunk Add-on for ServiceNow.

For information about this command, see Use custom streaming search commands for the Splunk Add-on for ServiceNow in Splunk Add-on for ServiceNow.

sort

Description

The `sort` command sorts all of the results by the specified fields. Results missing a given field are treated as having the smallest or largest possible value of that field if the order is descending or ascending, respectively.

If the first argument to the `sort` command is a number, then at most that many results are returned, in order. If no number is specified, the default limit of 10000 is used. If the number 0 is specified, all of the results are returned. See the `count` argument for more information.

Syntax

`sort [<count>] <sort-by-clause>... [desc]`

Required arguments

`<sort-by-clause>`

Syntax: `(- | +) <sort-field>, ( - | +) <sort-field> ...`

Description: List of fields to sort by and the sort order. Use a minus sign (-) for descending order and a plus sign (+) for ascending order. When
specifying more than one field, separate the field names with commas. See **Sort field options**.

**Optional arguments**

<count>

**Syntax:** <int>
**Description:** Specify the number of results to return from the sorted results. If no count is specified, the default limit of 10000 is used. If 0 is specified, all results are returned.
**Default:** 10000

desc

**Syntax:** d | desc
**Description:** Reverses the order of the results. If multiple fields are specified, reverses the order of the values in the fields in the order in which the fields are specified. For example, if there are three fields specified, the desc argument reverses the order of the values in the first field. For each set of duplicate values in the first field, reverses the order of the corresponding values in the second field. For each set of duplicate values in the second field, reverses the order of the corresponding values in the third field.

**Sort field options**

<sort-field>

**Syntax:** <field> | auto(<field>) | str(<field>) | ip(<field>) | num(<field>)
**Description:** Options you can specify with <sort-field>.

<field>

**Syntax:** <string>
**Description:** The name of field to sort.

auto

**Syntax:** auto(<field>)
**Description:** Determine automatically how to sort the values of the field.

ip

**Syntax:** ip(<field>)
**Description:** Interpret the values of the field as IP addresses.

num

**Syntax:** num(<field>)
**Description:** Interpret the values of the field as numbers.

\[ \text{str} \]

**Syntax:** \( \text{str(<field>)} \)

**Description:** Interpret the values of the field as strings and order the values alphabetically.

**Usage**

By default, \texttt{sort} tries to automatically determine what it is sorting. If the field takes on numeric values, the collating sequence is numeric. If the field takes on IP address values, the collating sequence is for IPs. Otherwise, the collating sequence is in lexicographical order. Some specific examples are:

- Alphabetic strings are sorted lexicographically.
- Punctuation strings are sorted lexicographically.
- Numeric data is sorted as you would expect for numbers and the sort order is specified as ascending or descending.
- Alphanumeric strings are sorted based on the data type of the first character. If the string starts with a number, the string is sorted numerically based on that number alone. Otherwise, strings are sorted lexicographically.
- Strings that are a combination of alphanumeric and punctuation characters are sorted the same way as alphanumeric strings.

In the default automatic mode for a field, the sort order is determined between each pair of values that are compared at any one time. This means that for some pairs of values, the order might be lexicographical, while for other pairs the order might be numerical. For example, if sorting in descending order: 10.1 > 9.1, but 10.1.a < 9.1.a.

**Lexicographical order**

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.

- Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically as 10, 100, 70, 9.
- Uppercase letters are sorted before lowercase letters.
Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

You can specify a custom sort order that overrides the lexicographical order. See the blog Order Up! Custom Sort Orders.

**Examples**

1. **Use the sort field options to specify field types**

Sort results by "ip" value in ascending order and then sort by the "url" value in descending order.

   ```
   ... | sort num(ip), -str(url)
   ```

2. **Specifying the number of results to sort**

Sort first 100 results in descending order of the "size" field and then by the "source" value in ascending order. This example specifies the type of data in each of the fields. The "size" field contains numbers and the "source" field contains strings.

   ```
   ... | sort 100 -num(size), +str(source)
   ```

3. **Specifying descending and ascending sort orders**

Sort results by the "_time" field in ascending order and then by the "host" value in descending order.

   ```
   ... | sort _time, -host
   ```

4. **Changing the time format of events for sorting**

Change the format of the event's time and sort the results in descending order by the Time field that is created with the `eval` command.

   ```
   ... | bin _time span=60m | eval Time=strftime(_time, "%m/%d %H:%M %Z")
   | stats avg(time_taken) AS AverageResponseTime BY Time | sort - Time
   ```

(Thanks to Splunk user Ayn for this example.)
5. **Specify a custom sort order**

Sort a table of results in a specific order, such as days of the week or months of the year, that is not lexicographical or numeric. For example, you have a search that produces the following table:

<table>
<thead>
<tr>
<th>Day</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friday</td>
<td>120</td>
</tr>
<tr>
<td>Monday</td>
<td>93</td>
</tr>
<tr>
<td>Tuesday</td>
<td>124</td>
</tr>
<tr>
<td>Thursday</td>
<td>356</td>
</tr>
<tr>
<td>Weekend</td>
<td>1022</td>
</tr>
<tr>
<td>Wednesday</td>
<td>248</td>
</tr>
</tbody>
</table>

Sorting on the day field (Day) returns a table sorted alphabetically, which does not make much sense. Instead, you want to sort the table by the day of the week, Monday to Friday. To do this, you first need to create a field (sort_field) that defines the order. Then you can sort on this field.

```
... | eval wd=lower(Day) | eval sort_field=case(wd="monday",1, wd="tuesday",2, wd="wednesday",3, wd="thursday",4, wd="friday",5, wd="weekend",6) | sort sort_field | fields - sort_field
```

This search uses the `eval` command to create the sort_field and the `fields` command to remove sort_field from the final results table.

(Thanks to Splunk users Ant1D and Ziegfried for this example.)

6. **Return the most recent event**

Return the most recent event:

```
... | sort 1 --time
```

See also

reverse

Answers

Have questions? Visit Splunk Answers and see what questions and answers the
Splunk community has used the sort command.

spath

Description

The spath command enables you to extract information from the structured data formats XML and JSON. The command stores this information in one or more fields. The command also highlights the syntax in the displayed events list.

You can also use the spath() function with the eval command. For more information, see the evaluation functions.

Syntax

spath [input=<field>] [output=<field>] [path=<datapath> | <datapath>]

Optional arguments

input

Syntax: input=<field>
Description: The field to read in and extract values.
Default: _raw

output

Syntax: output=<field>
Description: If specified, the value extracted from the path is written to this field name.
Default: If you do not specify an output argument, the value for the path argument becomes the field name for the extracted value.

path

Syntax: path=<datapath> | <datapath>
Description: The location path to the value that you want to extract. The location path can be specified as path=<datapath> or as just datapath. If you do not specify the path=, the first unlabeled argument is used as the location path. A location path is composed of one or more location steps, separated by periods. An example of this is 'foo.bar.baz'. A location step is composed of a field name and an optional index surrounded by curly brackets. The index can be an integer, to refer to the position of the data in an array (this differs between JSON and XML), or a string, to refer to an
XML attribute. If the index refers to an XML attribute, specify the attribute name with an @ symbol.

**Usage**

The spath command is a distributable streaming command. See [Command types](#).

**Location path omitted**

When used with no path argument, the spath command runs in "auto-extract" mode. In the "auto-extract" mode, the spath command finds and extracts all the fields from the first 5000 characters in the input field. These fields default to _raw if another input source is not specified. If a path is provided, the value of this path is extracted to a field named by the path or to a field specified by the output argument, if the output argument is provided.

**A location path contains one or more location steps**

A location path contains one or more location steps, each of which has a context that is specified by the location steps that precede it. The context for the top-level location step is implicitly the top-level node of the entire XML or JSON document.

**The location step is composed of a field name and an optional array index**

The location step is composed of a field name and an optional array index indicated by curly brackets around an integer or a string.

Array indices mean different things in XML and JSON. For example, JSON uses zero-based indexing. In JSON, foo.bar{3} refers to the fourth element of the bar child of the foo element. In XML, this same path refers to the third bar child of foo.

**Using wildcards in place of an array index**

The spath command lets you use wildcards to take the place of an array index in JSON. Now, you can use the location path entities.hashtags{}.text to get the text for all of the hashtags, as opposed to specifying entities.hashtags[0].text, entities.hashtags[1].text, and so on. The referenced path, here entities.hashtags, has to refer to an array for this to make sense. Otherwise, you get an error just like with regular array indices.
This also works with XML. For example, `catalog.book` and `catalog.book()` are equivalent. Both get you all the books in the catalog.

**Alternatives to the spath command**

If you are using autokv or index-time field extractions, the path extractions are performed for you at index time.

You do not need to explicitly use the `spath` command to provide a path.

If using `indexed_extractions=JSON` or using `KV_MODE=JSON` in the `props.conf` file, then the `spath` command is not necessary to explicitly use.

**Basic examples**

1. **Specify an output field and path**

This example shows how to specify a output field and path.

   ```
   ... | spath output=myfield path=foo.bar.baz
   ```

2. **Specify just the <datapath>**

   For the `path` argument, you can specify the location path with or without the `path=`. In this example the `<datapath>` is `server.name`.

   ```
   ... | spath output=myfield server.name
   ```

3. **Specify an output field and path based on an array**

   For example, you have this array.

   ```
   {
     "foo" : [1,2]
   }
   ```

   To specify the output field and path, use this syntax.

   ```
   ... | spath output=myfield path=foo(1)
   ```

4. **Specify an output field and a path that uses a nested array**

   For example, you have this nested array.
To specify the output and path from this nested array, use this syntax.

... | spath output=myfield path=foo.bar[].baz

5. **Specify the output field and a path for an XML attribute**

Use the @ symbol to specify an XML attribute. Consider the following XML list of books and authors.

```xml
<?xml version="1.0">
<purchases>
  <book>
    <author>Martin, George R.R.</author>
    <title yearPublished=1996>A Game of Thrones</title>
    <title yearPublished=1998>A Clash of Kings</title>
  </book>
  <book>
    <author>Clarke, Susanna</author>
    <title yearPublished=2004>Jonathan Strange and Mr. Norrell</title>
  </book>
  <book>
    <author>Kay, Guy Gavriel</author>
    <title yearPublished=1990>Tigana</title>
  </book>
  <book>
    <author>Bujold, Lois McMasters</author>
    <title yearPublished=1986>The Warrior's Apprentice</title>
  </book>
</purchases>
```

Use this search to return the path for the book and the year it was published.

... | spath output=dates path=purchases.book.title(@yearPublished) | table dates

In this example, the output is a single multivalue result that lists all of the years the books were published.
Extended examples

1: GitHub

As an administrator of a number of large Git repositories, you want to:

- See who has committed the most changes and to which repository
- Produce a list of the commits submitted for each user

Suppose you are Indexing JSON data using the GitHub PushEvent webhook. You can use the spath command to extract fields called repository, commit_author, and commit_id:

```bash
... | spath output=repository path=repository.url
... | spath output=commit_author path=commits{}.author.name
... | spath output=commit_id path=commits{}.id
```

To see who has committed the most changes to a repository, run the search.

```bash
... | top commit_author by repository
```

To see the list of commits by each user, run this search.

```bash
... | stats values(commit_id) by commit_author
```

2: Extract a subset of a XML attribute

This example shows how to extract values from XML attributes and elements.

```xml
<vendorProductSet vendorID="2">
  <product productID="17" units="mm">
    <prodName nameGroup="custom">
      <locName locale="all">APLI 01209</locName>
    </prodName>
    <desc descGroup="custom">
      <locDesc locale="es">Precios</locDesc>
      <locDesc locale="fr">Prix</locDesc>
      <locDesc locale="de">Preise</locDesc>
      <locDesc locale="ca">Preus</locDesc>
      <locDesc locale="pt">Preços</locDesc>
    </desc>
  </product>
</vendorProductSet>
```
To extract the values of the `locDesc` elements (Precios, Prix, Preise, etc.), use:

```
... | spath output=locDesc path=vendorProductSet.product.desc.locDesc
```

To extract the value of the `locale` attribute (es, fr, de, etc.), use:

```
... | spath output=locDesc.locale path=vendorProductSet.product.desc.locDesc[@locale]
```

To extract the attribute of the 4th `locDesc` (ca), use:

```
... | spath path=vendorProductSet.product.desc.locDesc{4}@locale
```

### 3: Extract and expand JSON events with multi-valued fields

The `mvexpand` command only works on one multivalued field. This example walks through how to expand a JSON event with more than one multivalued field into individual events for each field’s values. For example, given this event with `sourcetype=json`:

```json
{"widget": {
  "text": {
    "data": "Click here",
    "size": 36,
    "data": "Learn more",
    "size": 37,
    "data": "Help",
    "size": 38,
  }
}}
```

First, start with a search to extract the fields from the JSON. Because no `path` argument is specified, the `spath` command runs in "auto-extract" mode and extracts all of the fields from the first 5000 characters in the input field. The fields are then renamed and placed in a table.

```
sourcetype=json | spath | rename widget.text.size AS size, widget.text.data AS data | table _time,size,data
```

<table>
<thead>
<tr>
<th>_time</th>
<th>size</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-10-18 14:45:46.000 BST</td>
<td>36</td>
<td>Click here</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>Learn more</td>
</tr>
<tr>
<td></td>
<td>38</td>
<td>Help</td>
</tr>
</tbody>
</table>
Then, use the eval function, mvzip(), to create a new multivalued field named x, with the values of the size and data:

```
sourcetype=json | spath | rename widget.text.size AS size, widget.text.data AS data | eval x=mvzip(data,size) | table _time,data,size,x
```

<table>
<thead>
<tr>
<th>_time</th>
<th>data</th>
<th>size</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-10-18 14:45:46.000 BST</td>
<td>Click here</td>
<td>36</td>
<td>Click here,36</td>
</tr>
<tr>
<td></td>
<td>Learn more</td>
<td>37</td>
<td>Learn more,37</td>
</tr>
<tr>
<td></td>
<td>Help</td>
<td>38</td>
<td>Help,38</td>
</tr>
</tbody>
</table>

Now, use the mvexpand command to create individual events based on x and the eval function mvindex() to redefine the values for data and size.

```
sourcetype=json | spath | rename widget.text.size AS size, widget.text.data AS data | eval x=mvzip(data,size)| mvexpand x | eval x = split(x,,") | eval data=mvindex(x,0) | eval size=mvindex(x,1) | table _time,data, size
```

<table>
<thead>
<tr>
<th>_time</th>
<th>data</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-10-18 14:45:46.000 BST</td>
<td>Click here</td>
<td>36</td>
</tr>
<tr>
<td>2018-10-18 14:45:46.000 BST</td>
<td>Learn more</td>
<td>37</td>
</tr>
<tr>
<td>2018-10-18 14:45:46.000 BST</td>
<td>Help</td>
<td>38</td>
</tr>
</tbody>
</table>

(Thanks to Splunk user G. Zaimi for this example.)

**See also**

extract, kvform, multikv, regex, rex, xmlkv, xpath

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the spath command.

**stats**

591
Description

Calculates aggregate statistics, such as average, count, and sum, over the results set. This is similar to SQL aggregation. If the `stats` command is used without a `BY` clause, only one row is returned, which is the aggregation over the entire incoming result set. If a `BY` clause is used, one row is returned for each distinct value specified in the `BY` clause.

The `stats` command can be used for several SQL-like operations. If you are familiar with SQL but new to SPL, see Splunk SPL for SQL users.

**Difference between stats and eval commands**

The `stats` command calculates statistics based on fields in your events. The `eval` command creates new fields in your events by using existing fields and an arbitrary expression.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/18/15 6:05:56 PM</td>
<td>192.236.164.11 -- [18/JJ/15/18:20:36] GET /cart.do?action=addto/cartItemid=E2E5-14...</td>
</tr>
<tr>
<td>7/18/15 6:21:04 PM</td>
<td>192.236.164.11 -- [18/JJ/15/18:21:04] POST /addlink?itemID=E2E78&amp;E2E80%23=9009LIPF%20...</td>
</tr>
</tbody>
</table>

Result set before stats command

```bash
... | stats count(eval(method="GET")) as GET by host
```

<table>
<thead>
<tr>
<th>host</th>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>www1</td>
<td>8413</td>
</tr>
<tr>
<td>www2</td>
<td>4654</td>
</tr>
</tbody>
</table>

Result set after stats command
Syntax

Simple: \texttt{stats (stats-function(field) [AS field])... [BY field-list]}

Complete:

\begin{verbatim}
stats [partitions=<num>] [allnum=<bool>] [delim=<string>]
( <stats-agg-term>... | <sparkline-agg-term>... )
[<by-clause>]
[<dedup_splitvals>]
\end{verbatim}

Required arguments

\texttt{stats-agg-term}

\textbf{Syntax:} <stats-func>(<evaled-field> | <wc-field>) [AS <wc-field>]

\textbf{Description:} A statistical aggregation function. See \textit{Stats function options}. The function can be applied to an eval expression, or to a field or set of fields. Use the AS clause to place the result into a new field with a name that you specify. You can use wild card characters in field names. For more information on eval expressions, see Types of eval expressions in the \textit{Search Manual}.

\texttt{sparkline-agg-term}

\textbf{Syntax:} <sparkline-agg> [AS <wc-field>]

\textbf{Description:} A sparkline aggregation function. Use the AS clause to place the result into a new field with a name that you specify. You can use wild card characters in the field name.

Optional arguments

\texttt{allnum}

\textbf{Syntax:} allnum=<bool>

\textbf{Description:} If true, computes numerical statistics on each field if and only if all of the values of that field are numerical.

\textbf{Default:} false

\texttt{by-clause}

\textbf{Syntax:} BY <field-list>

\textbf{Description:} The name of one or more fields to group by. You cannot use a wildcard character to specify multiple fields with similar names. You must specify each field separately. The \texttt{BY} clause returns one row for each distinct value in the \texttt{BY} clause fields. If no \texttt{BY} clause is specified, the \texttt{stats} command returns only one row, which is the aggregation over the entire
incoming result set.

dedup_splitvals
  Syntax: dedup_splitvals=<boolean>
  Description: Specifies whether to remove duplicate values in multivalued
               BY clause fields.
  Default: false

delim
  Syntax: delim=<string>
  Description: Specifies how the values in the list() or values() aggregation
               are delimited.
  Default: a single space

partitions
  Syntax: partitions=<num>
  Description: If specified, partitions the input data based on the split-by
               fields for multithreaded reduce. The partitions argument runs the reduce
               step (in parallel reduce processing) with multiple threads in the same
               search process on the same machine. Compare that with parallel reduce,
               using the redistribute command, that runs the reduce step in parallel on
               multiple machines.
  Default: 1

Stats function options

stats-func
  Syntax: The syntax depends on the function that you use. Refer to the
  table below.
  Description: Statistical and charting functions that you can use with the
               stats command. Each time you invoke the stats command, you can use
               one or more functions. However, you can only use one BY clause. See
               Usage.

  The following table lists the supported functions by type of function. Use
  the links in the table to see descriptions and examples for each function.
  For an overview about using functions with commands, see Statistical and
  charting functions.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
</table>

594
<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aggregate functions</strong></td>
<td>avg()</td>
</tr>
<tr>
<td></td>
<td>count()</td>
</tr>
<tr>
<td></td>
<td>distinct_count()</td>
</tr>
<tr>
<td></td>
<td>estdc()</td>
</tr>
<tr>
<td></td>
<td>estdc_error()</td>
</tr>
<tr>
<td></td>
<td>exactperc&lt;int&gt;()</td>
</tr>
<tr>
<td></td>
<td>max()</td>
</tr>
<tr>
<td></td>
<td>median()</td>
</tr>
<tr>
<td></td>
<td>min()</td>
</tr>
<tr>
<td></td>
<td>mode()</td>
</tr>
<tr>
<td></td>
<td>perc&lt;int&gt;()</td>
</tr>
<tr>
<td></td>
<td>range()</td>
</tr>
<tr>
<td></td>
<td>stdev()</td>
</tr>
<tr>
<td></td>
<td>stdevp()</td>
</tr>
<tr>
<td></td>
<td>sum()</td>
</tr>
<tr>
<td></td>
<td>sumsq()</td>
</tr>
<tr>
<td></td>
<td>upperperc&lt;int&gt;()</td>
</tr>
<tr>
<td></td>
<td>var()</td>
</tr>
<tr>
<td></td>
<td>varp()</td>
</tr>
<tr>
<td><strong>Event order functions</strong></td>
<td>first()</td>
</tr>
<tr>
<td></td>
<td>last()</td>
</tr>
<tr>
<td><strong>Multivalue stats and chart functions</strong></td>
<td>list()</td>
</tr>
<tr>
<td></td>
<td>values()</td>
</tr>
<tr>
<td><strong>Time functions</strong></td>
<td>earliest()</td>
</tr>
<tr>
<td></td>
<td>latest()</td>
</tr>
<tr>
<td></td>
<td>rate()</td>
</tr>
<tr>
<td></td>
<td>earliest_time()</td>
</tr>
<tr>
<td></td>
<td>latest_time()</td>
</tr>
</tbody>
</table>

**Sparkline function options**

Sparklines are inline charts that appear within table cells in search results to display time-based trends associated with the primary key of each row. Read more about how to "Add sparklines to your search results" in the Search Manual.

**sparkline-agg**

*Syntax:* sparkline (count(<wc-field>), <span-length>) | sparkline (<sparkline-func>(<wc-field>), <span-length>)

*Description:* A sparkline specifier, which takes the first argument of an aggregation function on a field and an optional timespan specifier. If no timespan specifier is used, an appropriate timespan is chosen based on the time range of the search. If the sparkline is not scoped to a field, only the count aggregator is permitted. You can use wildcard characters in the field name. See the Usage section.

**sparkline-func**

*Syntax:* c() | count() | dc() | mean() | avg() | stdev() | stdevp() | var() | varp() | sum() | sumsq() | min() | max() | range()

*Description:* Aggregation function to use to generate sparkline values. Each sparkline value is produced by applying this aggregation to the
events that fall into each particular time bin.

Usage

The `stats` command is a **transforming command**. See Command types.

**Eval expressions with statistical functions**

When you use the `stats` command, you must specify either a statistical function or a sparkline function. When you use a statistical function, you can use an eval expression as part of the statistical function. For example:

```bash
index=* | stats count(eval(status="404")) AS count_status BY sourcetype
```

**Functions and memory usage**

Some functions are inherently more expensive, from a memory standpoint, than other functions. For example, the `distinct_count` function requires far more memory than the `count` function. The `values` and `list` functions also can consume a lot of memory.

If you are using the `distinct_count` function without a split-by field or with a low-cardinality split-by field, consider replacing the `distinct_count` function with the the `estdc` function (estimated distinct count). The `estdc` function might result in significantly lower memory usage and run times.

**Memory and maximum results**

In the `limits.conf` file, the `maxresultrows` setting in the `[searchresults]` stanza specifies the maximum number of results to return. The default value is 50,000. Increasing this limit can result in more memory usage.

The `max_mem_usage_mb` setting in the `[default]` stanza is used to limit how much memory the `stats` command uses to keep track of information. If the `stats` command reaches this limit, the command stops adding the requested fields to the search results. You can increase the limit, contingent on the available system memory.

If you are using Splunk Cloud and want to change either of these limits, file a Support ticket.
**Event order functions**

Using the `first` and `last` functions when searching based on time does not produce accurate results.

- To locate the first value based on time order, use the `earliest` function, instead of the `first` function.
- To locate the last value based on time order, use the `latest` function, instead of the `last` function.

For example, consider the following search.

```bash
index=test sourcetype=testDb | eventstats first(LastPass) as LastPass, last(_time) as mostRecentTestTime BY testCaseId | where startTime==LastPass OR _time==mostRecentTestTime | stats first(startTime) AS startTime, first(status) AS status, first(histID) AS currentHistId, last(histID) AS lastPassHistId BY testCaseId
```

Replace the `first` and `last` functions when you use the `stats` and `eventstats` commands for ordering events based on time. The following search shows the function changes.

```bash
index=test sourcetype=testDb | eventstats latest(LastPass) AS LastPass, earliest(_time) AS mostRecentTestTime BY testCaseId | where startTime==LastPass OR _time==mostRecentTestTime | stats latest(startTime) AS startTime, latest(status) AS status, latest(histID) AS currentHistId, earliest(histID) AS lastPassHistId BY testCaseId
```

**Wildcards in BY clauses**

The `stats` command does not support wildcard characters in field values in BY clauses.

For example, you cannot specify `| stats count BY source*`.

**Renaming fields**

You cannot rename one field with multiple names. For example if you have field A, you cannot rename A as B, A as C. The following example is not valid.

```bash
... | stats first(host) AS site, first(host) AS report
```
Basic Examples

1. Return the average transfer rate for each host

```
sourcetype=access* | stats avg(kbps) BY host
```

2. Search the access logs, and return the total number of hits from the top 100 values of "referer_domain"

Search the access logs, and return the total number of hits from the top 100 values of "referer_domain". The "top" command returns a count and percent value for each "referer_domain".

```
sourcetype=access_combined | top limit=100 referer_domain | stats sum(count) AS total
```

3. Calculate the average time for each hour for similar fields using wildcard characters

Return the average, for each hour, of any unique field that ends with the string "lay". For example, delay, xdelay, relay, etc.

```
... | stats avg(*lay) BY date_hour
```

4. Remove duplicates in the result set and return the total count for the unique results

Remove duplicates of results with the same "host" value and return the total count of the remaining results.

```
... | stats dc(host)
```

5. In a multivalue BY field, remove duplicate values

For each unique value of mvfield, return the average value of field. Deduplicates the values in the mvfield.

```
...| stats avg(field) BY mvfield dedup_splitvals=true
```

Extended Examples
1. Compare the difference between using the `stats` and `chart` commands

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

This search uses the `stats` command to count the number of events for a combination of HTTP status code values and host:

```
sourcetype=access_* | stats count BY status, host
```

The BY clause returns one row for each distinct value in the BY clause fields. In this search, because two fields are specified in the BY clause, every unique combination of status and host is listed on separate row.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>status</th>
<th>host</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>www1</td>
<td>11835</td>
</tr>
<tr>
<td>200</td>
<td>www2</td>
<td>11186</td>
</tr>
<tr>
<td>200</td>
<td>www3</td>
<td>11261</td>
</tr>
<tr>
<td>400</td>
<td>www1</td>
<td>233</td>
</tr>
<tr>
<td>400</td>
<td>www2</td>
<td>257</td>
</tr>
<tr>
<td>400</td>
<td>www3</td>
<td>211</td>
</tr>
<tr>
<td>403</td>
<td>www2</td>
<td>228</td>
</tr>
<tr>
<td>404</td>
<td>www1</td>
<td>244</td>
</tr>
<tr>
<td>404</td>
<td>www2</td>
<td>209</td>
</tr>
</tbody>
</table>

If you click the Visualization tab, the status field forms the X-axis and the host and count fields form the data series. The problem with this chart is that the host values (www1, www2, www3) are strings and cannot be measured in a chart.

Substitute the `chart` command for the `stats` command in the search.

```
sourcetype=access_* | chart count BY status, host
```

With the `chart` command, the two fields specified after the BY clause change the appearance of the results on the Statistics tab. The BY clause also makes the results suitable for displaying the results in a chart visualization.
• The first field you specify is referred to as the <row-split> field. In the table, the values in this field become the labels for each row. In the chart, this field forms the X-axis.
• The second field you specify is referred to as the <column-split> field. In the table, the values in this field are used as headings for each column. In the chart, this field forms the data series.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>status</th>
<th>www1</th>
<th>www2</th>
<th>www3</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>11835</td>
<td>11186</td>
<td>11261</td>
</tr>
<tr>
<td>400</td>
<td>233</td>
<td>257</td>
<td>211</td>
</tr>
<tr>
<td>403</td>
<td>0</td>
<td>288</td>
<td>0</td>
</tr>
<tr>
<td>404</td>
<td>244</td>
<td>209</td>
<td>237</td>
</tr>
<tr>
<td>406</td>
<td>258</td>
<td>228</td>
<td>224</td>
</tr>
<tr>
<td>408</td>
<td>267</td>
<td>243</td>
<td>246</td>
</tr>
<tr>
<td>500</td>
<td>225</td>
<td>262</td>
<td>246</td>
</tr>
<tr>
<td>503</td>
<td>324</td>
<td>299</td>
<td>329</td>
</tr>
<tr>
<td>505</td>
<td>242</td>
<td>0</td>
<td>238</td>
</tr>
</tbody>
</table>

If you click the Visualization tab, the status field forms the X-axis, the values in the host field form the data series, and the Y-axis shows the count.

2. Use eval expressions to count the different types of requests against each Web server

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

Run the following search to use the stats command to determine the number of different page requests, GET and POST, that occurred for each Web server.

```
sourcetype=access_* | stats count(eval(method="GET")) AS GET,
                     count(eval(method="POST")) AS POST BY host
```

This example uses eval expressions to specify the different field values for the stats command to count.
The first clause uses the `count()` function to count the Web access events that contain the `method` field value `GET`. Then, using the AS keyword, the field that represents these results is renamed `GET`.

The second clause does the same for POST events.

The counts of both types of events are then separated by the web server, using the BY clause with the `host` field.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>host</th>
<th>GET</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>www1</td>
<td>8431</td>
<td>5197</td>
</tr>
<tr>
<td>www2</td>
<td>8097</td>
<td>4815</td>
</tr>
<tr>
<td>www3</td>
<td>8338</td>
<td>4654</td>
</tr>
</tbody>
</table>

You can substitute the `chart` command for the `stats` command in this search. You can then click the Visualization tab to see a chart of the results.

### 3. Calculate a wide range of statistics by a specific field

**Count the number of earthquakes that occurred for each magnitude range**

This search uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the [USGS Earthquake Feeds](https://earthquake.usgs.gov/颏a/) and upload the file to your Splunk instance. This example uses the *All Earthquakes* data from the past 30 days.

Run the following search to calculate the number of earthquakes that occurred in each magnitude range. This data set is comprised of events over a 30-day period.

```
source=all_month.csv | chart count AS "Number of Earthquakes" BY mag span=1 | rename mag AS "Magnitude Range"
```

- **This search uses `span=1`** to define each of the ranges for the magnitude field, `mag`.
- **The `rename command` is then used to rename the field to "Magnitude Range".**

---

601
The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>Magnitude Range</th>
<th>Number of Earthquakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1-0</td>
<td>18</td>
</tr>
<tr>
<td>0-1</td>
<td>2088</td>
</tr>
<tr>
<td>1-2</td>
<td>3005</td>
</tr>
<tr>
<td>2-3</td>
<td>1026</td>
</tr>
<tr>
<td>3-4</td>
<td>194</td>
</tr>
<tr>
<td>4-5</td>
<td>452</td>
</tr>
<tr>
<td>5-4</td>
<td>109</td>
</tr>
<tr>
<td>6-7</td>
<td>11</td>
</tr>
<tr>
<td>7-8</td>
<td>3</td>
</tr>
</tbody>
</table>

Click the Visualization tab to see the result in a chart.

Calculate aggregate statistics for the magnitudes of earthquakes in an area

Search for earthquakes in and around California. Calculate the number of earthquakes that were recorded. Use statistical functions to calculate the minimum, maximum, range (the difference between the min and max), and average magnitudes of the recent earthquakes. List the values by magnitude type.

```
source=all_month.csv place=*California* | stats count, max(mag), min(mag), range(mag), avg(mag) BY magType
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>magType</th>
<th>count</th>
<th>max(mag)</th>
<th>min(mag)</th>
<th>range(mag)</th>
<th>avg(mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>123</td>
<td>2.8</td>
<td>0.0</td>
<td>2.8</td>
<td>0.549593</td>
</tr>
<tr>
<td>MbLg</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000000</td>
</tr>
<tr>
<td>Md</td>
<td>1565</td>
<td>3.2</td>
<td>0.1</td>
<td>3.1</td>
<td>1.056486</td>
</tr>
<tr>
<td>Me</td>
<td>2</td>
<td>2.0</td>
<td>1.6</td>
<td>.04</td>
<td>1.800000</td>
</tr>
<tr>
<td>Ml</td>
<td>1202</td>
<td>4.3</td>
<td>-0.4</td>
<td>4.7</td>
<td>1.226622</td>
</tr>
<tr>
<td>Mw</td>
<td>6</td>
<td>4.9</td>
<td>3.0</td>
<td>1.9</td>
<td>3.650000</td>
</tr>
</tbody>
</table>

602
Find the mean, standard deviation, and variance of the magnitudes of the recent quakes

Search for earthquakes in and around California. Calculate the number of earthquakes that were recorded. Use statistical functions to calculate the mean, standard deviation, and variance of the magnitudes for recent earthquakes. List the values by magnitude type.

source=usgs place=*California* | stats count mean(mag), stdev(mag), var(mag) BY magType

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>magType</th>
<th>count</th>
<th>mean(mag)</th>
<th>std(mag)</th>
<th>var(mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>123</td>
<td>0.549593</td>
<td>0.356985</td>
<td>0.127438</td>
</tr>
<tr>
<td>MbLg</td>
<td>1</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>Md</td>
<td>1565</td>
<td>1.056486</td>
<td>0.580042</td>
<td>0.336449</td>
</tr>
<tr>
<td>Me</td>
<td>2</td>
<td>1.800000</td>
<td>0.346410</td>
<td>0.120000</td>
</tr>
<tr>
<td>Ml</td>
<td>1202</td>
<td>1.226622</td>
<td>0.629664</td>
<td>0.396476</td>
</tr>
<tr>
<td>Mw</td>
<td>6</td>
<td>3.650000</td>
<td>0.716240</td>
<td>0.513000</td>
</tr>
<tr>
<td>ml</td>
<td>10</td>
<td>0.934000</td>
<td>0.560401</td>
<td>0.314049</td>
</tr>
</tbody>
</table>

The mean values should be exactly the same as the values calculated using \( avg() \).

4. In a table display items sold by ID, type, and name and calculate the revenue for each product

This example uses the sample dataset from the Search Tutorial and a field lookup to add more information to the event data.

- Download the data set from Add data tutorial and follow the instructions to load the tutorial data.
- Download the CSV file from Use field lookups tutorial and follow the instructions to set up the lookup definition to add price and productName to the events.
After you configure the field lookup, you can run this search using the time range, **All time**.

Create a table that displays the items sold at the Buttercup Games online store by their ID, type, and name. Also, calculate the revenue for each product.

```
sourcetype=access_* status=200 action=purchase | stats values(categoryId) AS Type, values(productName) AS "Product Name", sum(price) AS "Revenue" by productId | rename productId AS "Product ID" | eval Revenue="$ ".tostring(Revenue,"commas")
```

This example uses the ```values()``` function to display the corresponding ```categoryId``` and ```productName``` values for each ```productId```. Then, it uses the ```sum()``` function to calculate a running total of the values of the ```price``` field.

Also, this example renames the various fields, for better display. For the ```stats``` functions, the renames are done inline with an “AS” clause. The ```rename``` command is used to change the name of the ```product_id``` field, since the syntax does not let you rename a split-by field.

Finally, the results are piped into an ```eval``` expression to reformat the ```Revenue``` field values so that they read as currency, with a dollar sign and commas.

This returns the following table of results:

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Type</th>
<th>Product Name</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR-AG-001</td>
<td>ARCADE</td>
<td>Design Space Retro</td>
<td>$5346.66</td>
</tr>
<tr>
<td>CL-RG-016</td>
<td>SPORTS</td>
<td>Century 2014</td>
<td>$2786.42</td>
</tr>
<tr>
<td>ER-SS-007</td>
<td>STRATEGY</td>
<td>Medieval Kingdoms</td>
<td>$25676.02</td>
</tr>
<tr>
<td>EC-SS-002</td>
<td>STRATEGY</td>
<td>Dream Clash</td>
<td>$5273.04</td>
</tr>
<tr>
<td>IR-AG-001</td>
<td>ARCADE</td>
<td>Dixieland Notes</td>
<td>$1856.35</td>
</tr>
<tr>
<td>PI-SS-007</td>
<td>STRATEGY</td>
<td>First Nation</td>
<td>$4599.30</td>
</tr>
<tr>
<td>MS-AG-007</td>
<td>ARCADE</td>
<td>Margarita Bros.</td>
<td>$926.79</td>
</tr>
</tbody>
</table>

5. **Determine how much email comes from each domain**

This example uses sample email data. You should be able to run this search on any email data by replacing the ```sourcetype=cisco:esa``` with the ```sourcetype``` value and the ```mailfrom``` field with email address field name in your data. For example, the email might be To, From, or CC.

Find out how much of the email in your organization comes from .com, .net, .org or other top level domains.

The ```eval``` command in this search contains two expressions, separated by a comma.
The first part of this search uses the eval command to break up the email address in the mailfrom field. The from_domain is defined as the portion of the mailfrom field after the @ symbol.

- The split() function is used to break the mailfrom field into a multivalue field called accountname. The first value of accountname is everything before the "@" symbol, and the second value is everything after.
- The mvindex() function is used to set from_domain to the second value in the multivalue field accountname.

The results are then piped into the stats command. The count() function is used to count the results of the eval expression.

The eval uses the match() function to compare the from_domain to a regular expression that looks for the different suffixes in the domain. If the value of from_domain matches the regular expression, the count is updated for each suffix, .com, .net, and .org. Other domain suffixes are counted as other.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>.com</th>
<th>.net</th>
<th>.org</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>4246</td>
<td>9890</td>
<td>0</td>
<td>3543</td>
</tr>
</tbody>
</table>

6. Search Web access logs for the total number of hits from the top 10 referring domains

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

This example searches the web access logs and return the total number of hits from the top 10 referring domains.
This search uses the `top` command to find the ten most common referer domains, which are values of the `referer` field. Some events might use `referer_domain` instead of `referer`. The `top` command returns a count and percent value for each referer.

You can then use the `stats` command to calculate a total for the top 10 referrer accesses.

```
sourcetype=access_*  | top limit=10 referer  | stats sum(count) AS total
```

The `sum()` function adds the values in the `count` to produce the total number of times the top 10 referrers accessed the web site.

**See also**

**Commands**
- eventstats, rare, sistats, streamstats, top

**Blogs**
- Getting started with stats, eventstats and streamstats
- Search commands > stats, chart, and timechart
- Smooth operator | Searching for multiple field values

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the stats command.
**strcat**

**Description**

Concatenates string values from 2 or more fields. Combines together string values and literals into a new field. A destination field name is specified at the end of the `strcat` command.

**Syntax**

`strcat [allrequired=<bool>] <source-fields> <dest-field>`

**Required arguments**

**<dest-field>**

*Syntax:* `<string>`

*Description:* A destination field to save the concatenated string values in, as defined by the `<source-fields>` argument. The destination field is always at the end of the series of source fields.

**<source-fields>**

*Syntax:* `(<field> | <quoted-str>)...`

*Description:* Specify the field names and literal string values that you want to concatenate. Literal values must be enclosed in quotation marks.

**quoted-str**

*Syntax:* `"<string>"`

*Description:* Quoted string literals.

*Examples:* `"/"` or `"."`

**Optional arguments**

**allrequired**

*Syntax:* `allrequired=<bool>`

*Description:* Specifies whether or not all source fields need to exist in each event before values are written to the destination field. If `allrequired=f`, the destination field is always written and source fields that do not exist are treated as empty strings. If `allrequired=t`, the values are written to destination field only if all source fields exist.

*Default:* false
Usage

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

Examples

**Example 1:**

Add a field called `comboIP`, which combines the source and destination IP addresses. Separate the addresses with a forward slash character.

```bash
... | strcat sourceIP "/" destIP comboIP
```

**Example 2:**

Add a field called `comboIP`, which combines the source and destination IP addresses. Separate the addresses with a forward slash character. Create a chart of the number of occurrences of the field values.

```bash
host="mailserver" | strcat sourceIP "/" destIP comboIP | chart count by comboIP
```

**Example 3:**

Add a field called `address`, which combines the host and port values into the format `<host>::<port>`.

```bash
... | strcat host "::" port address
```

See also

eval

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `strcat` command.
streamstats

Description

Adds cumulative summary statistics to all search results in a streaming manner. The `streamstats` command calculates statistics for each event at the time the event is seen. For example, you can calculate the running total for a particular field. The total is calculated by using the values in the specified field for every event that has been processed, up to the current event.

Syntax

```
streamstats [reset_on_change=<bool>] [reset_before="("<eval-expression>")"]
[reset_after="("<eval-expression>")"] [current=<bool>] [window=<int>]
[time_window=<span-length>] [global=<bool>] [allnum=<bool>]
<stats-agg-term>... [<by-clause>]
```

**Required arguments**

*stats-agg-term*

- **Syntax:** `<stats-func>( <evaled-field> | <wc-field> ) [AS <wc-field>]`
- **Description:** A statistical aggregation function. See Stats function options. The function can be applied to an eval expression, or to a field or set of fields. Use the AS clause to place the result into a new field with a name that you specify. You can use wild card characters in field names. For more information on eval expressions, see Types of eval expressions in the Search Manual.

**Optional arguments**

*allnum*

- **Syntax:** `allnum=<boolean>`
- **Description:** If true, computes numerical statistics on each field only if all of the values in that field are numerical.
- **Default:** false

*by-clause*

- **Syntax:** `BY <field-list>`
- **Description:** The name of one or more fields to group by.

*current*

- **Syntax:** `current=<boolean>`
**Description:** If true, the search includes the given, or current, event in the summary calculations. If false, the search uses the field value from the previous event.

**Default:** true

**global**

**Syntax:** global=<boolean>

**Description:** Used only when the window argument is set. Defines whether to use a single window, global=true, or to use separate windows based on the by clause. If global=false and window is set to a non-zero value, a separate window is used for each group of values of the field specified in the by clause.

**Default:** true

**reset_after**

**Syntax:** reset_after="(<eval-expression>)"

**Description:** After the streamstats calculations are produced for an event, reset_after specifies that all of the accumulated statistics are reset if the eval-expression returns true. The eval-expression must evaluate to true or false. The eval-expression can reference fields that are returned by the streamstats command. When the reset_after argument is combined with the window argument, the window is also reset when the accumulated statistics are reset.

**Default:** false

**reset_before**

**Syntax:** reset_before="(<eval-expression>)"

**Description:** Before the streamstats calculations are produced for an event, reset_before specifies that all of the accumulated statistics are reset when the eval-expression returns true. The eval-expression must evaluate to true or false. When the reset_before argument is combined with the window argument, the window is also reset when the accumulated statistics are reset.

**Default:** false

**reset_on_change**

**Syntax:** reset_on_change=<bool>

**Description:** Specifies that all of the accumulated statistics are reset when the group by fields change. The reset is as if no previous events have been seen. Only events that have all of the group by fields can trigger a reset. Events that have only some of the group by fields are ignored. The eval-expression must evaluate to true or false. When the reset_on_change argument is combined with the window argument, the
window is also reset when the accumulated statistics are reset. See the Usage section.
Default: false

time_window
Syntax: time_window=<span-length>
Description: Specifies the window size for the streamstats calculations, based on time. The time_window argument is limited by range of values in the _time field in the events. To use the time_window argument, the events must be sorted in either ascending or descending time order. You can use the window argument with the time_window argument to specify the maximum number of events in a window. For the <span-length>, to specify five minutes, use time_window=5m. To specify 2 days, use time_window=2d.
Default: None. However, the value of the max_stream_window attribute in the limits.conf file applies. The default value is 10000 events.

window
Syntax: window=<integer>
Description: Specifies the number of events to use when computing the statistics.
Default: 0, which means that all previous and current events are used.

Stats function options

stats-func
Syntax: The syntax depends on the function that you use. Refer to the table below.
Description: Statistical and charting functions that you can use with the streamstats command. Each time you invoke the streamstats command, you can use one or more functions. However, you can only use one BY clause. See Usage.

The following table lists the supported functions by type of function. Use the links in the table to see descriptions and examples for each function. For an overview about using functions with commands, see Statistical and charting functions.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg()</td>
</tr>
</tbody>
</table>

611
<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate functions</td>
<td>count()</td>
</tr>
<tr>
<td></td>
<td>distinct_count()</td>
</tr>
<tr>
<td></td>
<td>estdc()</td>
</tr>
<tr>
<td></td>
<td>estdc_error()</td>
</tr>
<tr>
<td></td>
<td>max()</td>
</tr>
<tr>
<td></td>
<td>median()</td>
</tr>
<tr>
<td></td>
<td>min()</td>
</tr>
<tr>
<td></td>
<td>mode()</td>
</tr>
<tr>
<td></td>
<td>range()</td>
</tr>
<tr>
<td></td>
<td>stdev()</td>
</tr>
<tr>
<td></td>
<td>stdevp()</td>
</tr>
<tr>
<td></td>
<td>sumsq()</td>
</tr>
<tr>
<td></td>
<td>upperperc&lt;int&gt;()</td>
</tr>
<tr>
<td></td>
<td>var()</td>
</tr>
<tr>
<td></td>
<td>varp()</td>
</tr>
<tr>
<td>Event order functions</td>
<td>earliest()</td>
</tr>
<tr>
<td></td>
<td>first()</td>
</tr>
<tr>
<td></td>
<td>last()</td>
</tr>
<tr>
<td></td>
<td>latest()</td>
</tr>
<tr>
<td>Multivalue stats and chart</td>
<td>list(X)</td>
</tr>
<tr>
<td>functions</td>
<td>values(X)</td>
</tr>
</tbody>
</table>

### Usage

The `streamstats` command is a centralized streaming command. See Command types.

The `streamstats` command is similar to the `eventstats` command except that it uses events before the current event to compute the aggregate statistics that are applied to each event. If you want to include the current event in the statistical calculations, use `current=true`, which is the default.

The `streamstats` command is also similar to the `stats` command in that `streamstats` calculates summary statistics on search results. Unlike `stats`, which works on the group of results as a whole, `streamstats` calculates statistics for each event at the time the event is seen.

**Escaping string values**

If your `<eval-expression>` contains a value instead of a field name, you must escape the quotation marks around the value.

The following example is a simple way to see this. Start by using the `makeresults` command to create 3 events. Use the `streamstats` command to produce a cumulative count of the events. Then use the `eval` command to create a simple test. If the value of the `count` field is equal to 2, display `yes` in the `test` field. Otherwise display `no` in the `test` field.
The results appear something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>count</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-01-11 11:32:43</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>2017-01-11 11:32:43</td>
<td>2</td>
<td>yes</td>
</tr>
<tr>
<td>2017-01-11 11:32:43</td>
<td>3</td>
<td>no</td>
</tr>
</tbody>
</table>

Use the `streamstats` command to reset the count when the match is true. You must escape the quotation marks around the word `yes`. The following example shows the complete search.

```
| makeresults count=3 | streamstats count | eval test=if(count==2,"yes","no") | streamstats count as testCount reset_after="("match(test,\"yes\")")"
```

Here is another example. You want to look for the value `session is closed` in the `description` field. Because the value is a string, you must enclose it in quotation marks. You then need to escape those quotation marks.

```
... | streamstats reset_after="("description=="session is closed")")"
```

**The `reset_on_change` argument**

You have a dataset with the field "shift" that contains either the value DAY or the value NIGHT. You run this search:

```
...| streamstats count BY shift reset_on_change=true
```

If the dataset is:

```
shift
DAY
DAY
NIGHT
NIGHT
NIGHT
NIGHT
DAY
NIGHT
```
Running the command with `reset_on_change=true` produces the following streamstats results:

<table>
<thead>
<tr>
<th>shift, count</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY, 1</td>
</tr>
<tr>
<td>DAY, 2</td>
</tr>
<tr>
<td>NIGHT, 1</td>
</tr>
<tr>
<td>NIGHT, 2</td>
</tr>
<tr>
<td>NIGHT, 3</td>
</tr>
<tr>
<td>NIGHT, 4</td>
</tr>
<tr>
<td>DAY, 1</td>
</tr>
<tr>
<td>NIGHT, 1</td>
</tr>
</tbody>
</table>

**Basic examples**

**1. Calculate a snapshot of summary statistics**

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range **All time** when you run the search.

You want to determine the number of the bytes used over a set period of time. The following search uses the first 5 events. Because search results typically display the most recent event first, the `sort` command is used to sort the 5 events in ascending order to see the oldest event first and the most recent event last. Ascending order enables the `streamstats` command to calculate statistics over time.

```
sourcetype=access_combined* | head 5 | sort _time
```
Add the `streamstats` command to the search to generate a running total of the bytes over the 5 events and organize the results by `clientip`.

```
sourcetype=access_combined* | head 5 | sort _time | streamstats sum(bytes) AS ASimpleSumOfBytes BY clientip
```

When you click on the `ASimpleSumOfBytes` field in the list of `Interesting fields`, an information window shows the cumulative sum of the bytes, as shown in this image:
The `streamstats` command aggregates the statistics to the original data, which means that all of the original data is accessible for further calculations.

Add the `table` command to the search to display the only the values in the `_time`, `clientip`, `bytes`, and `ASimpleSumOfBytes` fields.

```
sourcetype=access_combined* | head 5 | sort _time | streamstats sum(bytes) as ASimpleSumOfBytes by clientip | table _time, clientip, bytes, ASimpleSumOfBytes
```

Each event shows the timestamp for the event, the clientip, and the number of bytes used. The `ASimpleSumOfBytes` field shows a cumulative summary of the bytes for each clientip.

### 2. Compute the average of a field over the last 5 events

For each event, compute the average of field `foo` over the last 5 events, including the current event. Similar to doing trendline `sma5(foo)`

```
... | streamstats avg(foo) window=5
```

### 3. Compute the average of a field, with a by clause, over the last 5 events

For each event, compute the average value of `foo` for each value of `bar` including only 5 events, specified by the window size, with that value of `bar`.

```
... | streamstats avg(foo) by bar window=5 global=f
```

### 4. For each event, add a count of the number of events processed

This example adds to each event a count field that represents the number of events seen so far, including that event. For example, it adds 1 for the first event, 2 for the second event, and so on.

```
... | streamstats count
```
If you did not want to include the current event, you would specify:

```
... | streamstats count current=f
```

**5. Apply a time-based window to streamstats**

Assume that the `max_stream_window` argument in the `limits.conf` file is the default value of 10000 events.

The following search counts the events, using a time window of five minutes.

```
... | streamstats count time_window=5m
```

This search adds a count field to each event.

- If the events are in descending time order (most recent to oldest), the value in the count field represents the number of events in the next 5 minutes.
- If the events are in ascending time order (oldest to most recent), the count field represents the number of events in the previous 5 minutes.

If there are more events in the time-based window than the value for the `max_stream_window` argument, the `max_stream_window` argument takes precedence. The count will never be > 10000, even if there are actually more than 10,000 events in any 5 minute period.

**Extended examples**

**6. Calculate the running total of distinct users over time**

Each day you track unique users, and you would like to track the cumulative count of distinct users. This example calculates the running total of distinct users over time.

```
eventtype="download" | bin _time span=1d as day | stats values(clientip) as ips dc(clientip) by day | streamstats dc(ips) as "Cumulative total"
```

The `bin` command breaks the time into days. The `stats` command calculates the distinct users (clientip) and user count per day. The `streamstats` command finds the running distinct count of users.

This search returns a table that includes: `day`, `ips`, `dc(clientip)`, and `Cumulative total`. 

617
7. Calculate hourly cumulative totals

This example uses `streamstats` to produce hourly cumulative totals.

```
... | timechart span=1h sum(bytes) as SumOfBytes | streamstats global=f sum(*) as accu_total_*
```

This search returns 3 columns: _time, SumOfBytes, and accu_total_SumOfBytes.

The `timechart` command buckets the events into spans of 1 hour and counts the total values for each category. The `timechart` command also fills NULL values, so that there are no missing values. Then, the `streamstats` command is used to calculate the accumulated total.

This example uses `streamstats` to produce hourly cumulative totals for category values.

```
... | timechart span=1h sum(value) as total by category | streamstats global=f | addtotals | accum Total | rename Total as accu_total
```

8. Calculate when a DHCP IP lease address changed for a specific MAC address

This example uses `streamstats` to figure out when a DHCP IP lease address changed for a MAC address, 54:00:00:00:00:00.

```
source=dhcp MAC=54:00:00:00:00:00 | head 10 | streamstats current=f last(DHCP_IP) as new_dhcp_ip last(_time) as time_of_change by MAC
```

You can also clean up the presentation to display a table of the DHCP IP address changes and the times the occurred.

```
source=dhcp MAC=54:00:00:00:00:00:00 | head 10 | streamstats current=f last(DHCP_IP) as new_dhcp_ip last(_time) as time_of_change by MAC where DHCP_IP!=new_dhcp_ip | convert ctime(time_of_change) as time_of_change | rename DHCP_IP as old_dhcp_ip | table time_of_change, MAC, old_dhcp_ip, new_dhcp_ip
```

For more details, refer to the Splunk Blogs post for this example.
See also

Commands
accum, autoregress, delta, fillnull, eventstats, trendline

Blogs
Getting started with stats, eventstats and streamstats

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the streamstats command.

table

Description

The `table` command returns a table that is formed by only the fields that you specify in the arguments. Columns are displayed in the same order that fields are specified. Column headers are the field names. Rows are the field values. Each row represents an event.

The `table` command is similar to the `fields` command in that it lets you specify the fields you want to keep in your results. Use `table` command when you want to retain data in tabular format.

With the exception of a scatter plot to show trends in the relationships between discrete values of your data, you should not use the `table` command for charts. See Usage.

Syntax

table <wc-field-list>

Arguments

<wc-field-list>

**Syntax:** <wc-field> <wc-field> ...
**Description:** A list of field names. You can use wild card characters in the field names.
Usage

The `table` command is a **transforming command**. See [Command types](#).

Visualizations

Other than a scatter plot, you should not use the `table` command for visualizations. Splunk Web requires the internal fields, which are the fields that begin with an underscore character, to render the visualizations. The `table` command strips these fields out of the results by default. To build visualizations, you should use the `fields` command instead. The `fields` command always retains all the internal fields.

Command type

The `table` command is a non-streaming command. If you are looking for a streaming command similar to the `table` command, use the `fields` command.

Field renaming

The `table` command doesn't let you rename fields, only specify the fields that you want to show in your tabulated results. If you're going to rename a field, do it before piping the results to `table`.

Truncated results

The `table` command truncates the number of results returned based on settings in the `limits.conf` file. In the `[search]` stanza, if the value for the `truncate_report` parameter is 1, the number of results returned is truncated.

The number of results is controlled by the `max_count` parameter in the `[search]` stanza. If `truncate_report` is set to 0, the `max_count` parameter is not applied.

Examples

Example 1

This example uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.
You can download a current CSV file from the **USGS Earthquake Feeds** and add it as an input to your Splunk deployment.

Search for recent earthquakes in and around California and display only the time of the quake (*DateTime*), where it occurred (*Region*), and the quake’s magnitude (*Magnitude*) and depth (*Depth*).

```
index=usgs_* source=usgs place=*California | table time, place, mag, depth
```

This simply reformats your events into a table and displays only the fields that you specified as arguments.

### Example 2

This example uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the **USGS Earthquake Feeds** and add it as an input to your Splunk deployment.

Show the date, time, coordinates, and magnitude of each recent earthquake in Northern California.

```
index=usgs_* source=usgs place=*California | rename lat as latitude lon as longitude | table time, place, lat*, lon*, mag
```

This example begins with a search for all recent earthquakes in Northern California (**Region="Northern California"**).

Then it pipes these events into the `rename` command to change the names of the coordinate fields, from `lat` and `lon` to `latitude` and `longitude`. (The `table` command doesn't let you rename or reformat fields, only specify the fields that you want to show in your tabulated results.)
Finally, it pipes the results into the `table` command and specifies both coordinate fields with `lat*`, `lon*`, the magnitude with `mag`, and the date and time with `time`.

This example just illustrates how the `table` command syntax allows you to specify multiple fields using the asterisk wildcard.

**Example 3**

This example uses the sample dataset from the tutorial but should work with any format of Apache Web access log. Download the data set from the Add data tutorial and follow the instructions to get the sample data into your Splunk deployment. Then, run this search using the time range, **All time**.

Search for IP addresses and classify the network they belong to.

```
sourcetype=access_* | dedup clientip | eval network=if(cidrmatch("192.0.0.0/16", clientip), "local", "other") | table clientip, network
```

This example searches for Web access data and uses the `dedup` command to remove duplicate values of the IP addresses (`clientip`) that access the server. These results are piped into the `eval` command, which uses the `cidrmatch()` function to compare the IP addresses to a subnet range (192.0.0.0/16). This search also uses the `if()` function, which says that if the value of `clientip` falls in the subnet range, then `network` is given the value `local`. Otherwise, `network=other`.

The results are then piped into the `table` command to show only the distinct IP addresses (`clientip`) and the network classification (`network`):
More examples

Example 1: Create a table for fields foo, bar, then all fields that start with 'baz'.

... | table foo bar baz*

See Also

fields

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the table command.

tags

Description

Annotates specified fields in your search results with tags. If there are fields specified, only annotate tags for those fields. Otherwise, look for tags for all fields. If outputfield is specified, the tags for all fields will be written to this field. If outputfield is specified, inclname and inclvalue control whether or not the field name and field values are added to the output field. By default only the tag itself is written to the outputfield, that is (<field>::)?(<value>::)?tag .

Syntax

tags [outputfield=<field>] [inclname=<bool>] [inclvalue=<bool>] <field-list>

Required arguments

<field-list>

Syntax: <field> <field> ...

Description: Specify the fields to annotate with tags.

Optional arguments

outputfield

Syntax: outputfield=<field>
**Description:** If specified, the tags for all fields will be written to this field. Otherwise, the tags for each field will be written to a field named `tag::<field>.

inclname

**Syntax:** inclname=T|F  
**Description:** If outputfield is specified, controls whether or not the field name is added to the output field.  
**Default:** F

inclvalue

**Syntax:** inclvalue=T|F  
**Description:** If outputfield is specified, controls whether or not the field value is added to the output field.  
**Default:** F

Usage

The `tags` command is a distributable streaming command. See [Command types](#).

Examples

**Example 1:**
Write tags for host and eventtype fields into tag::host and tag::eventtype.

```
... | tags host eventtype
```

**Example 2:**
Write new field test that contains tags for all fields.

```
... | tags outputfield=test
```

**Example 3:**
Write tags for host and sourcetype into field test in the format host::<tag> or sourcetype::<tag>.

```
... | tags outputfield=test inclname=t host sourcetype
```
See also

eval

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the tags command.

tail

Description

Returns the last N number of specified results. The events are returned in reverse order, starting at the end of the result set. The last 10 events are returned if no integer is specified.

Syntax

tail [<N>]

Required arguments

None.

Optional arguments

<N>

Syntax: <int>
Description: The number of results to return.
Default: 10

Examples

Example 1:

Return the last 20 results in reverse order.

... | tail 20
See also
head, reverse

Answers
Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the tail command.

timechart

Description
Creates a time series chart with corresponding table of statistics.

A timechart is a statistical aggregation applied to a field to produce a chart, with time used as the X-axis. You can specify a split-by field, where each distinct value of the split-by field becomes a series in the chart. If you use an eval expression, the split-by clause is required. With the limit and agg options, you can specify series filtering. These options are ignored if you specify an explicit where-clause. If you set limit=0, no series filtering occurs.

Syntax

timechart [sep=<string>] [format=<string>] [partial=<bool>] [cont=<bool>] [limit=<int>] [agg=<stats-agg-term>] [<bin-options>... ] ( ( <single-agg> [BY <split-by-clause>] ) | ( <eval-expression> ) BY <split-by-clause> ) [ <dedup_splitvals> ]

Required arguments
When specifying timechart command arguments, either <single-agg> or <eval-expression> BY <split-by-clause> is required.

eval-expression

Syntax: <math-exp> | <concat-exp> | <compare-exp> | <bool-exp> | <function-call>

Description: A combination of literals, fields, operators, and functions that represent the value of your destination field. For these evaluations to
work, your values need to be valid for the type of operation. For example, with the exception of addition, arithmetic operations might not produce valid results if the values are not numerical. Additionally, the search can concatenate the two operands if they are both strings. When concatenating values with a period \( . \) the search treats both values as strings, regardless of their actual data type.

**single-agg**

**Syntax:** count | <stats-func>(<field>)

**Description:** A single aggregation applied to a single field, including an evaluated field. For \(<stats-func>\), see Stats function options. No wildcards are allowed. The field must be specified, except when using the count function, which applies to events as a whole.

**split-by-clause**

**Syntax:** <field> (<tc-options>)... [where-clause]

**Description:** Specifies a field to split the results by. If field is numerical, default discretization is applied. Discretization is defined with the tc-options. Use the where-clause to specify the number of columns to include. See the tc options and the where clause sections in this topic.

**Optional arguments**

**agg=<stats-agg-term>**

**Syntax:** agg=( <stats-func> ( <evaled-field> | <wc-field> ) [AS <wc-field>] )

**Description:** A statistical aggregation function. See Stats function options. The function can be applied to an eval expression, or to a field or set of fields. Use the AS clause to place the result into a new field with a name that you specify. You can use wild card characters in field names.

**bin-options**

**Syntax:** bins | minspan | span | <start-end> | aligntime

**Description:** Options that you can use to specify discreet bins, or groups, to organize the information. The bin-options set the maximum number of bins, not the target number of bins. See the Bin options section in this topic.

**Default:** bins=100

**cont**

**Syntax:** cont=<bool>

**Description:** Specifies whether the chart is continuous or not. If set to true, the Search application fills in the time gaps.

**Default:** true
dedup_splitvals

Syntax: dedup_splitvals=<boolean>
Description: Specifies whether to remove duplicate values in multivalued <split-by-clause> fields.
Default: false

fixedrange

Syntax: fixedrange=<bool>
Description: Specify whether or not to enforce the earliest and latest times of the search. Setting fixedrange=false allows the timechart command to constrict to just the time range with valid data.
Default: true

format

Syntax: format=<string>
Description: Used to construct output field names when multiple data series are used in conjunction with a split-by-field. format takes precedence over sep and allows you to specify a parameterized expression with the stats aggregator and function ($AGG$) and the value of the split-by-field ($VAL$).

limit

Syntax: limit=<int>
Description: Specifies a limit for the number of distinct values of the split-by field to return. If set to limit=0, all distinct values are used. Setting limit=N keeps the N highest scoring distinct values of the split-by field. All other values are grouped into 'OTHER', as long as useother is not set to false. The scoring is determined as follows:
   ◊ If a single aggregation is specified, the score is based on the sum of the values in the aggregation for that split-by value. For example, for timechart avg(foo) BY <field> the avg(foo) values are added up for each value of <field> to determine the scores.
   ◊ If multiple aggregations are specified, the score is based on the frequency of each value of <field>. For example, for timechart avg(foo) max(bar) BY <field>, the top scoring values for <field> are the most common values of <field>.

Ties in scoring are broken lexicographically, based on the value of the split-by field. For example, 'BAR' takes precedence over 'bar', which takes precedence over 'foo'. See Usage.

partial

Syntax: partial=<bool>
**Description:** Controls if partial time bins should be retained or not. Only the first and last bin can be partial.

**Default:** True. Partial time bins are retained.

`sep`

**Syntax:** `sep=<string>`

**Description:** Used to construct output field names when multiple data series are used in conjunctions with a split-by field. This is equivalent to setting `format` to `$AGG$<sep>$VAL$`.

**Stats function options**

`stats-func`

**Syntax:** The syntax depends on the function that you use. Refer to the table below.

**Description:** Statistical functions that you can use with the `timechart` command. Each time you invoke the `timechart` command, you can use one or more functions. However, you can only use one `BY` clause. See `Usage`.

The following table lists the supported functions by type of function. Use the links in the table to see descriptions and examples for each function. For an overview about using functions with commands, see Statistical and charting functions.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aggregate functions</strong></td>
<td><code>avg()</code> <code>count()</code> <code>distinct_count()</code> <code>estdc()</code> <code>estdc_error()</code> <code>exactperc&lt;int&gt;()</code> <code>perc&lt;int&gt;()</code> <code>sum()</code> <code>sumsq()</code> <code>upperperc&lt;int&gt;()</code> <code>var()</code> <code>varp()</code></td>
</tr>
<tr>
<td>Event order functions</td>
<td><code>earliest()</code> <code>first()</code> <code>last()</code> <code>latest()</code></td>
</tr>
<tr>
<td>Multivalue stats and chart functions</td>
<td><code>list(X)</code> <code>values(X)</code></td>
</tr>
</tbody>
</table>
### Type of function  
<table>
<thead>
<tr>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time functions</td>
</tr>
<tr>
<td>per_day()</td>
</tr>
<tr>
<td>per_hour()</td>
</tr>
<tr>
<td>per_minute()</td>
</tr>
<tr>
<td>per_second()</td>
</tr>
</tbody>
</table>

**Bin options**

**bins**  
**Syntax:** `bins=<int>`  
**Description:** Sets the *maximum number* of bins to discretize into. This does not set the target number of bins. It finds the smallest bin size that results in no more than \( N \) distinct bins. Even though you specify a number such as 300, the resulting number of bins might be much lower.  
**Default:** 100

**minspan**  
**Syntax:** `minspan=<span-length>`  
**Description:** Specifies the smallest span granularity to use automatically inferring span from the data time range. See *Usage*.

**span**  
**Syntax:** `span=<log-span> | span=<span-length> | span=<snap-to-time>`  
**Description:** Sets the size of each bin, using either a log-based span, a span length based on time, or a span that snaps to a specific time. For descriptions of each of these options, see *Span options*.

The starting time of a bin might not match your local timezone. see *Usage*.

**<start-end>**  
**Syntax:** `end=<num> | start=<num>`  
**Description:** Sets the minimum and maximum extents for numerical bins. Data outside of the \([\text{start}, \text{end}]\) range is discarded.

**align/time**  
**Syntax:** `align/time=(earliest | latest | <time-specifier>)`  
**Description:** Align the bin times to something other than base UNIX time (epoch 0). The `align/time` option is valid only when doing a time-based discretization. Ignored if `span` is in days, months, or years.
Span options

$log-span$
Syntax: \([\text{<num>}]\log[\text{<num>}]\)
Description: Sets to log-based span. The first number is a coefficient. The second number is the base. If the first number is supplied, it must be a real number \(\geq 1.0\) and \(<\) base. Base, if supplied, must be real number \(> 1.0\) (strictly greater than 1).

<span-length>
Syntax: \(<\text{int}>[\text{<timescale}>]\)
Description: A span of each bin, based on time. If the timescale is provided, this is used as a time range. If not, this is an absolute bin length.

<timescale>
Syntax: \(<\text{sec}>\mid <\text{min}>\mid <\text{hr}>\mid <\text{day}>\mid <\text{week}>\mid <\text{month}>\mid <\text{subseconds}>\)
Description: Timescale units.
Default: \(<\text{sec}>\)

<table>
<thead>
<tr>
<th>Time scale</th>
<th>Valid syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sec&gt;</td>
<td>s \mid \text{sec} \mid \text{secs} \mid second \mid seconds</td>
<td>Time scale in seconds.</td>
</tr>
<tr>
<td>&lt;min&gt;</td>
<td>m \mid \text{min} \mid \text{mins} \mid minute \mid minutes</td>
<td>Time scale in minutes.</td>
</tr>
<tr>
<td>&lt;hr&gt;</td>
<td>h \mid \text{hr} \mid \text{hrs} \mid hour \mid hours</td>
<td>Time scale in hours.</td>
</tr>
<tr>
<td>&lt;day&gt;</td>
<td>d \mid \text{day} \mid days</td>
<td>Time scale in days.</td>
</tr>
<tr>
<td>&lt;month&gt;</td>
<td>mon \mid \text{month} \mid months</td>
<td>Time scale in months.</td>
</tr>
<tr>
<td>&lt;subseconds&gt;</td>
<td>us \mid ms \mid cs \mid ds</td>
<td>Time scale in microseconds (us), milliseconds (ms), centiseconds (cs), or deciseconds (ds)</td>
</tr>
</tbody>
</table>

<snap-to-time>
Syntax: \([+\mid-] [\text{<time_integer}>] <\text{relative_time_unit}>@<\text{snap_to_time_unit}>\)
Description: A span of each bin, based on a relative time unit and a snap to time unit. The <snap-to-time> must include a relative_time_unit, the @ symbol, and a snap_to_time_unit. The offset, represented by the plus (+)
or minus (-) is optional. If the <time_integer> is not specified, 1 is the default. For example if you specify w as the relative_time_unit, 1 week is assumed. The option is used only with week time units. It cannot be used with other time units such as minutes or quarters.

**tc options**

The <tc-option> is part of the <split-by-clause>.

tc-option  
**Syntax:** <bin-options> | usenull=<bool> | useother=<bool> | nullstr=<string> | otherstr=<string>  
**Description:** Timechart options for controlling the behavior of splitting by a field.

**bin-options**

See the Bin options section in this topic.

**nullstr**

**Syntax:** nullstr=<string>  
**Description:** If usenull=true, specifies the label for the series that is created for events that do not contain the split-by field.  
**Default:** NULL

**otherstr**

**Syntax:** otherstr=<string>  
**Description:** If useother=true, specifies the label for the series that is created in the table and the graph.  
**Default:** OTHER

**usenull**

**Syntax:** usenull=<bool>  
**Description:** Controls whether or not a series is created for events that do not contain the split-by field. The label for the series is controlled by the nullstr option.  
**Default:** true

**useother**

**Syntax:** useother=<bool>  
**Description:** You specify which series to include in the results table by using the <agg>, <limit>, and <where-clause> options. The useother
option specifies whether to merge all of the series not included in the results table into a single new series. If \texttt{useother=true}, the label for the series is controlled by the \texttt{otherstr} option.

\textbf{Default:} true

\textit{where clause}

The \texttt{<where-clause>} is part of the \texttt{<split-by-clause>}. The \texttt{<where-clause>} is comprised of two parts, a single aggregation and \textit{See Where clause examples}. where clause

\textbf{Syntax:} \texttt{<single-agg> \textless where-comp>}

\textbf{Description:} Specifies the criteria for including particular data series when a field is given in the \texttt{<tc-by-clause>}. The most common use of this option is to look for spikes in your data rather than overall mass of distribution in series selection. The default value finds the top ten series by area under the curve. Alternately one could replace sum with max to find the series with the ten highest spikes. Essentially the default is the same as specifying \texttt{where sum in top10}. The \texttt{<where-clause>} has no relation to the code\texttt{where} command.

\texttt{<where-comp>}

\textbf{Syntax:} \texttt{<wherein-comp> | <wherethresh-comp>}

\textbf{Description:} Specify either a grouping for the series or the threshold for the series.

\texttt{<wherein-comp>}

\textbf{Syntax:} \texttt{(in | notin) (top | bottom)<int>}

\textbf{Description:} A grouping criteria that requires the aggregated series value be in or not in some top or bottom group.

\texttt{<wherethresh-comp>}

\textbf{Syntax:} \texttt{(< | >) [" "] <num>}

\textbf{Description:} A threshold criteria that requires the aggregated series value be greater than or less than some numeric threshold. You can specify the threshold with or without a space between the sign and the number.

\textbf{Usage}

The \texttt{timechart} command is a \textbf{transforming command}. See \texttt{Command types}.
**bins and span arguments**

The `timechart` command accepts either the `bins` argument OR the `span` argument. If you specify both `bins` and `span`, `span` is used. The `bins` argument is ignored.

If you do not specify either `bins` or `span`, the `timechart` command uses the default `bins=100`.

**Default time spans**

If you use the predefined time ranges in the time range picker, and do not specify the `span` argument, the following table shows the default span that is used.

<table>
<thead>
<tr>
<th>Time range</th>
<th>Default span</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last 15 minutes</td>
<td>10 seconds</td>
</tr>
<tr>
<td>Last 60 minutes</td>
<td>1 minute</td>
</tr>
<tr>
<td>Last 4 hours</td>
<td>5 minutes</td>
</tr>
<tr>
<td>Last 24 hours</td>
<td>30 minutes</td>
</tr>
<tr>
<td>Last 7 days</td>
<td>1 day</td>
</tr>
<tr>
<td>Last 30 days</td>
<td>1 day</td>
</tr>
<tr>
<td>Previous year</td>
<td>1 month</td>
</tr>
</tbody>
</table>

(Thanks to Splunk users MuS and Martin Mueller for their help in compiling this default time span information.)

**Spans used when minspan is specified**

When you specify a `minspan` value, the span that is used for the search must be equal to or greater than one of the span threshold values in the following table. For example, if you specify `minspan=15m` that is equivalent to 900 seconds. The minimum span that can be used is 1800 seconds, or 30 minutes.

<table>
<thead>
<tr>
<th>Span threshold</th>
<th>Time equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 second</td>
<td></td>
</tr>
<tr>
<td>5 seconds</td>
<td></td>
</tr>
<tr>
<td>10 seconds</td>
<td></td>
</tr>
<tr>
<td>30 seconds</td>
<td></td>
</tr>
<tr>
<td>Span threshold</td>
<td>Time equivalents</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>60 seconds</td>
<td>1 minute</td>
</tr>
<tr>
<td>300 seconds</td>
<td>5 minutes</td>
</tr>
<tr>
<td>600 seconds</td>
<td>10 minutes</td>
</tr>
<tr>
<td>1800 seconds</td>
<td>30 minutes</td>
</tr>
<tr>
<td>3600 seconds</td>
<td>1 hour</td>
</tr>
<tr>
<td>86400 seconds</td>
<td>1 day</td>
</tr>
<tr>
<td>2592000 seconds</td>
<td>30 days</td>
</tr>
</tbody>
</table>

**Bin time spans and local time**

The span option always rounds down the starting date for the first bin. There is no guarantee that the bin start time used by the timechart command corresponds to your local timezone. In part this is due to differences in daylight savings time for different locales. To use day boundaries, use span=1d. Do not use not span=86400s, or span=1440m, or span=24h.

**Bin time spans versus per_\* functions**

The functions, `per_day()`, `per_hour()`, `per_minute()`, and `per_second()` are aggregator functions and are not responsible for setting a time span for the resultant chart. These functions are used to get a consistent scale for the data when an explicit span is not provided. The resulting span can depend on the search time range.

For example, `per_hour()` converts the field value so that it is a rate per hour, or `sum()/(<hours in the span>)`. If your chart span ends up being 30m, it is `sum(*)^2`.

If you want the span to be 1h, you still have to specify the argument `span=1h` in your search.

**Note:** You can do `per_hour()` on one field and `per_minute()` (or any combination of the functions) on a different field in the same search.

**Split-by fields**

If you specify a split-by field, ensure that you specify the `bins` and `span` arguments before the split-by field. If you specify these arguments after the split-by field, Splunk software assumes that you want to control the bins on the split-by field, not on the time axis.
If you use `chart` or `timechart`, you cannot use a field that you specify in a function as your split-by field as well. For example, you will not be able to run:

```plaintext
... | chart sum(A) by A span=log2
```

However, you can work around this with an `eval` expression, for example:

```plaintext
... | eval A1=A | chart sum(A) by A1 span=log2
```

**Functions and memory usage**

Some functions are inherently more expensive, from a memory standpoint, than other functions. For example, the `distinct_count` function requires far more memory than the `count` function. The `values` and `list` functions also can consume a lot of memory.

If you are using the `distinct_count` function without a split-by field or with a low-cardinality split-by by field, consider replacing the `distinct_count` function with the `estdc` function (estimated distinct count). The `estdc` function might result in significantly lower memory usage and run times.

**Lexicographical order**

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.

- Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically as 10, 100, 70, 9.
- Uppercase letters are sorted before lowercase letters.
- Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

You can specify a custom sort order that overrides the lexicographical order. See the blog Order Up! Custom Sort Orders.

**Basic Examples**

1. Chart the product of the average "CPU" and average "MEM" for each "host"
For each minute, compute the product of the average "CPU" and average "MEM" for each "host".

```
... | timechart span=1m eval(avg(CPU) * avg(MEM)) BY host
```

**2. Chart the average of cpu_seconds by processor**

Create a timechart of the average of cpu_seconds by processor, rounded to 2 decimal places.

```
... | timechart eval(round(avg(cpu_seconds),2)) BY processor
```

**3. Chart the average of "CPU" for each "host"**

For each minute, calculate the average value of "CPU" for each "host".

```
... | timechart span=1m avg(CPU) BY host
```

**4. Chart the average "cpu_seconds" by "host" and remove outlier values**

Calculate the average "cpu_seconds" by "host". Remove outlying values that might distort the timechart axis.

```
... | timechart avg(cpu_seconds) BY host | outlier action=tf
```

**5. Chart the average "thruput" of hosts over time**

```
... | timechart span=5m avg(thruput) BY host
```

**6. Chart the eventypes by source_ip**

For each minute, count the eventypes by source_ip, where the count is greater than 10.

```
sshd failed OR failure | timechart span=1m count(eventtype) BY source_ip
usenull=f WHERE count>10
```

**7. Align the chart time bins to local time**

Align the time bins to 5am (local time). Set the span to 12h. The bins will represent 5am - 5pm, then 5pm - 5am (the next day), and so on.

```
...| timechart _time span=12h aligntime=@d+5h
```
8. In a multivalue BY field, remove duplicate values

For each unique value of mvfield, return the average value of field. Deduplicates the values in the mvfield.

...| timechart avg(field) BY mvfield dedup_splitval=true

Extended Examples

1. Chart revenue for the different products

This example uses the sample dataset from the Search Tutorial and a field lookup to add more information to the event data. To try this example for yourself:

- Download the tutorialdata.zip file from this topic in the Search Tutorial and follow the instructions to upload the file to your Splunk deployment.
- Download the Prices.csv.zip file from this topic in the Search Tutorial and follow the instructions to set up your field lookup.
- Use the time range Yesterday when you run the search.

The tutorialdata.zip file includes a productId field that is the catalog number for the items sold at the Buttercup Games online store. The field lookup uses the prices.csv file to add two new fields to your events: productName, which is a descriptive name for the item, and price, which is the cost of the item.

Chart the revenue for the different products that were purchased yesterday.

sourcetype=access_* action=purchase | timechart per_hour(price) by productName usenull=f useother=f

- This example searches for all purchase events (defined by the action=purchase).
- The results are piped into timechart command.
- The per_hour() function sums up the values of the price field for each productName and organizes the total by time.

This search produces the following table of results in the Statistics tab. To format the numbers to the proper digits for currency, click the format icon in the column heading. On the Number Formatting tab, select the Precision.
After you create this chart, you can position your mouse pointer over each section to view more metrics for the product purchased at that hour of the day.

Notice that the chart does not display the data in hourly spans. Because a span is not provided (such as span=1hr), the per_hour() function converts the value so that it is a sum per hours in the time range (which in this example is 24 hours).

2. Chart daily purchases by product type

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

Chart the number of purchases made daily for each type of product.

```
sourcetype=access_* action=purchase | timechart span=1d count by categoryId usenull=f
```

- This example searches for all purchases events, defined by the action=purchase, and pipes those results into the timechart command.
• The `span=1day` argument buckets the count of purchases over the week into daily chunks.
• The `usenull=f` argument ignore any events that contain a NULL value for `categoryId`.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>ACCESSORIES</th>
<th>ARCADE</th>
<th>SHOOTER</th>
<th>SIMULATION</th>
<th>SPORTS</th>
<th>STRATEGY</th>
<th>TEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-03-29</td>
<td>5</td>
<td>17</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>32</td>
<td>9</td>
</tr>
<tr>
<td>2018-03-30</td>
<td>62</td>
<td>63</td>
<td>39</td>
<td>30</td>
<td>22</td>
<td>127</td>
<td>56</td>
</tr>
<tr>
<td>2018-03-31</td>
<td>65</td>
<td>94</td>
<td>38</td>
<td>42</td>
<td>34</td>
<td>128</td>
<td>60</td>
</tr>
<tr>
<td>2018-04-01</td>
<td>54</td>
<td>82</td>
<td>42</td>
<td>39</td>
<td>13</td>
<td>115</td>
<td>66</td>
</tr>
<tr>
<td>2018-04-02</td>
<td>52</td>
<td>63</td>
<td>45</td>
<td>42</td>
<td>22</td>
<td>124</td>
<td>52</td>
</tr>
<tr>
<td>2018-04-03</td>
<td>46</td>
<td>76</td>
<td>34</td>
<td>42</td>
<td>19</td>
<td>123</td>
<td>59</td>
</tr>
<tr>
<td>2018-04-04</td>
<td>57</td>
<td>70</td>
<td>36</td>
<td>38</td>
<td>20</td>
<td>130</td>
<td>56</td>
</tr>
<tr>
<td>2018-04-05</td>
<td>46</td>
<td>72</td>
<td>35</td>
<td>37</td>
<td>13</td>
<td>106</td>
<td>46</td>
</tr>
</tbody>
</table>

Click the Visualization tab. If necessary, change the chart to a column chart.

Compare the number of different items purchased each day and over the course of the week.

3. Display results in 1 week intervals

This search uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.

You can download a current CSV file from the USGS Earthquake Feeds and upload the file to your Splunk instance. This example uses the All Earthquakes data from the past 30 days.
This search counts the number of earthquakes in Alaska where the magnitude is greater than or equal to 3.5. The results are organized in spans of 1 week, where the week begins on Monday.

```
source=all_month.csv place=*alaska* mag>=3.5 | timechart span=w@w1 count BY mag
```

- The <by-clause> is used to group the earthquakes by magnitude.
- You can only use week spans with the snap-to span argument in the `timechart` command. For more information, see Specify a snap to time unit.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>3.5</th>
<th>3.6</th>
<th>3.7</th>
<th>3.8</th>
<th>4</th>
<th>4.1</th>
<th>4.1</th>
<th>4.3</th>
<th>4.4</th>
<th>4.5</th>
<th>OTHER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-03-26</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2018-04-02</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2018-04-09</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2018-04-16</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2018-04-23</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

4. **Count the revenue for each item over time**

This example uses the sample dataset from the Search Tutorial and a field lookup to add more information to the event data. Before you run this example:

- Download the data set from this topic in the Search Tutorial and follow the instructions to upload it to your Splunk deployment.
- Download the Prices.csv.zip file from this topic in the Search Tutorial and follow the instructions to set up your field lookup.

The original data set includes a `productId` field that is the catalog number for the items sold at the Buttercup Games online store. The field lookup adds two new fields to your events: `productName`, which is a descriptive name for the item, and `price`, which is the cost of the item.

Count the total revenue made for each item sold at the shop over the **last 7 days**. This example shows two different searches to generate the calculations.

**Search 1**

The first search uses the `span` argument to bucket the times of the search results into 1 day increments. The search then uses the `sum()` function to
add the **price** for each **product_name**.

```
sourcetype=access_* action=purchase | timechart span=1d sum(price) by productName usenull=f
```

**Search 2**

This second search uses the `per_day()` function to calculate the total of the **price** values for each day.

```
sourcetype=access_* action=purchase | timechart per_day(price) by productName usenull=f
```

Both searches produce similar results. Search 1 produces values with two decimal places. Search 2 produces values with six decimal places. The following image shows the results from Search 1.

Click the **Visualization** tab. If necessary, change the chart to a column chart.

Now you can compare the total revenue made for items purchased each day and over the course of the week.

**5. Chart product views and purchases for a single day**

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to **get the tutorial data into Splunk**. Use the time range **Yesterday** when you run the search.
Chart a single day's views and purchases at the Buttercup Games online store.

`sourcetype=access_* | timechart per_hour(eval(method="GET")) AS Views, per_hour(eval(action="purchase")) AS Purchases`

- This search uses the `per_hour()` function and `eval` expressions to search for page views (`method=GET`) and purchases (`action=purchase`).
- The results of the `eval` expressions are renamed as Views and Purchases, respectively.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>_time</th>
<th>Views</th>
<th>Purchases</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-04-05 00:00:00</td>
<td>150.000000</td>
<td>44.000000</td>
</tr>
<tr>
<td>2018-04-05 00:30:00</td>
<td>166.000000</td>
<td>54.000000</td>
</tr>
<tr>
<td>2018-04-05 01:00:00</td>
<td>214.000000</td>
<td>72.000000</td>
</tr>
<tr>
<td>2018-04-05 01:30:00</td>
<td>242.000000</td>
<td>80.000000</td>
</tr>
<tr>
<td>2018-04-05 02:00:00</td>
<td>158.000000</td>
<td>26.000000</td>
</tr>
<tr>
<td>2018-04-05 02:30:00</td>
<td>166.000000</td>
<td>20.000000</td>
</tr>
<tr>
<td>2018-04-05 03:00:00</td>
<td>220.000000</td>
<td>56.000000</td>
</tr>
</tbody>
</table>

Click the **Visualization** tab. Format the results as an area chart.

The difference between the two areas indicates that many of the views did not become purchases. If all of the views became purchases, you would expect the areas to overlay on top each other completely. There would be no difference between the two areas.
Where clause examples

These examples use the `where` clause to control the number of series values returned in the time-series chart.

**Example 1:** Show the 5 most rare series based on the minimum count values. All other series values will be labeled as "other".

```
index=_internal | timechart span=1h count by source WHERE min in bottom5
```

**Example 2:** Show the 5 most frequent series based on the maximum values. All other series values will be labeled as "other".

```
index=_internal | timechart span=1h count by source WHERE max in top5
```

These two searches return six data series: the five top or bottom series specified and the series labeled other. To hide the "other" series, specify the argument `useother=f`.

**Example 3:** Show the source series count of INFO events, but only where the total number of events is larger than 100. All other series values will be labeled as "other".

```
index=_internal | timechart span=1h
sum(eval(if(log_level=="INFO",1,0))) by source WHERE sum > 100
```

**Example 4:** Using the where clause with the count function measures the total number of events over the period. This yields results similar to using the sum function.

The following two searches returns the sources series with a total count of events greater than 100. All other series values will be labeled as "other".

```
index=_internal | timechart span=1h count by source WHERE count > 100
index=_internal | timechart span=1h count by source WHERE sum > 100
```

**See also**

Commands

**bin**
Search commands > stats, chart, and timechart

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the timechart command.

**timewrap**

**Description**

Displays, or wraps, the output of the timechart command so that every period of time is a different series.

You can use the timewrap command to compare data over specific time period, such as day-over-day or month-over-month. You can also use the timewrap command to compare multiple time periods, such as a two week period over another two week period. See Timescale options.

**Syntax**

```
timewrap <timewrap-span> [align=now | end] [series=relative | exact | short] [time_format=<str>]
```

**Required arguments**

**timewrap-span**

**Syntax:** `[<int>]<timescale>

**Description:** A span of each bin, based on time. The `timescale` is required. The `int` is not required. If `<int>` is not specified, 1 is assumed. For example if `day` is specified for the timescale, `1day` is assumed. See Timescale options.

**Optional arguments**

**align**

**Syntax:** `align=now | end`
**Description:** Specifies if the wrapping should be aligned to the current time or the end time of the search.

**Default:** end

**series**

**Syntax:** series=relative | exact | short

**Description:** Specifies how the data series is named. If `series=relative` and `timewrap-span` is set to week, the field names are latest_week, 1week_before, 2weeks_before, and so forth. If `series=exact`, use the `time_format` argument to specify a custom format for the series names.

**Default:** relative

**time_format**

**Syntax:** time_format=<str>

**Description:** Use with `series=exact` to specify a custom name for the series. The `time_format` is designed to be used with the `time format` variables. For example, if you specify `time_format="week of %d/%m/%y"`, this format appears as week of 13/2/17 and week of 20/2/17. If you specify `time_format=week of %b %d`, this format appears as week of Feb 13 and week of Feb 20. See the **Usage** section.

**Default:** None

**Timescale options**

**<timescale>**

**Syntax:** <sec> | <min> | <hr> | <day> | <week> | <month> | <quarter> | <year>

**Description:** Time scale units.

<table>
<thead>
<tr>
<th>Time scale</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sec&gt;</td>
<td>s</td>
<td>sec</td>
</tr>
<tr>
<td>&lt;min&gt;</td>
<td>min</td>
<td>mins</td>
</tr>
<tr>
<td>&lt;hr&gt;</td>
<td>h</td>
<td>hr</td>
</tr>
<tr>
<td>&lt;day&gt;</td>
<td>d</td>
<td>day</td>
</tr>
<tr>
<td>&lt;week&gt;</td>
<td>w</td>
<td>week</td>
</tr>
<tr>
<td>&lt;month&gt;</td>
<td>m</td>
<td>mon</td>
</tr>
<tr>
<td>&lt;quarter&gt;</td>
<td>qtr</td>
<td>quarter</td>
</tr>
<tr>
<td>&lt;year&gt;</td>
<td>y</td>
<td>yr</td>
</tr>
</tbody>
</table>
The `timewrap` command uses the abbreviation *m* to refer to months. Other commands, such as `timechart` and `bin` use the abbreviation *m* to refer to minutes.

**Usage**

The `timewrap` command is a reporting command.

You must use the `timechart` command in the search before you use the `timewrap` command.

The wrapping is based on the end time of the search. If you specify the time range of *All time*, the wrapping is based on today's date. You see this in the timestamps for the `_time` field and in the data series names.

**Using the `time_format` argument**

If the format you specify does not contain any time specifiers, then all of the data series display the same name and are compressed into each other.

**Basic example**

Display a timechart that has a span of 1 day for each count in a week over week comparison table. Each table column, which is the series, is 1 week of time.

```
... | timechart count span=1d | timewrap 1week
```

**Extended example**

**See also**

timechart

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the timewrap command.
**top**

**Description**

Finds the most common values for the fields in the field list. Calculates a count and a percentage of the frequency the values occur in the events. If the <by-clause> is included, the results are grouped by the field you specify in the <by-clause>.

**Syntax**

top [<N>] [top-options]... <field-list> [by-clause]

**Required arguments**

*<field-list>*

**Syntax:** <field>, <field>, ...

**Description:** Comma-delimited list of field names.

**Optional arguments**

*<N>*

**Syntax:** <int>

**Description:** The number of results to return.

**Default:** 10

*<top-options>*

**Syntax:** countfield=<string> | limit=<int> | otherstr=<string> | percentfield=<string> | showcount=<bool> | showperc=<bool> | useother=<bool>

**Description:** Options for the top command. See Top options.

*<by-clause>*

**Syntax:** BY <field-list>

**Description:** The name of one or more fields to group by.

**Top options**

countfield

**Syntax:** countfield=<string>

**Description:** For each value returned by the top command, the results also return a count of the events that have that value. This argument
specifies the name of the field that contains the count. The count is returned by default. If you do not want to return the count of events, specify showcount=false. 
**Default:** count

**limit**

**Syntax:** limit=<int>  
**Description:** Specifies how many results to return. To return all values, specify zero (0). Specifying `top limit=<int>` is the same as specifying `top N`.  
**Default:** 10

**otherstr**

**Syntax:** otherstr=<string>  
**Description:** If useother=true, a row representing all other values is added to the results. Use `otherstr=<string>` to specify the name of the label for the row.  
**Default:** OTHER

**percentfield**

**Syntax:** percentfield=<string>  
**Description:** For each value returned by the `top` command, the results also return a percentage of the events that have that value. This argument specifies the name of the field that contains the percentage. The percentage is returned by default. If you do not want to return the percentage of events, specify showperc=false.  
**Default:** percent

**showcount**

**Syntax:** showcount=<bool>  
**Description:** Specify whether to create a field called "count" (see "countfield" option) with the count of that tuple.  
**Default:** true

**showperc**

**Syntax:** showperc=<bool>  
**Description:** Specify whether to create a field called "percent" (see "percentfield" option) with the relative prevalence of that tuple.  
**Default:** true

**useother**

**Syntax:** useother=<bool>
**Description:** Specify whether or not to add a row that represents all values not included due to the limit cutoff.

**Default:** false

**Usage**

The `top` command is a **transforming command**. See [Command types](#).

**Default fields**

When you use the `top` command, two fields are added to the results: `count` and `percent`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>count</code></td>
<td>The number of events in your search results that contain the field values that are returned by the <code>top</code> command. See the <code>countfield</code> and <code>showcount</code> arguments.</td>
</tr>
<tr>
<td><code>percent</code></td>
<td>The percentage of events in your search results that contain the field values that are returned by the <code>top</code> command. See the <code>percentfield</code> and <code>showperc</code> arguments.</td>
</tr>
</tbody>
</table>

**Default maximum number of results**

By default the `top` command returns a maximum of 50,000 results. This maximum is controlled by the `maxresultrows` setting in the `[top]` stanza in the `limits.conf` file. Increasing this limit can result in more memory usage.

Only users with file system access, such as system administrators, can edit the configuration files. Never change or copy the configuration files in the `default` directory. The files in the `default` directory must remain intact and in their original location. Make the changes in the `local` directory.

See [How to edit a configuration file](#).

If you are using Splunk Cloud and want to edit the configuration file, file a Support ticket.

**Examples**
Example 1: Return the 20 most common values for a field

This search returns the 20 most common values of the "referer" field. The results show the number of events (count) that have that a count of referer, and the percent that each referer is of the total number of events.

```
sourcetype=access_* | top limit=20 referer
```

![Table of most common referers](image)

Example 2: Return top values for one field organized by another field

This search returns the top "action" values for each "referer_domain".

```
sourcetype=access_* | top action by referer_domain
```

Because a limit is not specified, this returns all the combinations of values for "action" and "referer_domain" as well as the counts and percentages:

![Table of top actions by referer_domain](image)

Example 3: Returns the top product purchased for each category

This example uses the sample dataset from the Search Tutorial and a field lookup to add more information to the event data.

- Download the data set from Add data tutorial and follow the instructions to load the tutorial data.
- Download the CSV file from Use field lookups tutorial and follow the instructions to set up the lookup definition to add price and productName.

651
to the events.

After you configure the field lookup, you can run this search using the time range, **All time**.

This search returns the top product purchased for each category. Do not show the percent field. Rename the count field to "total".

```
sourcetype=access_* status=200 action=purchase | top 1 productName by categoryId showperc=f countfield=total
```

### See also

rare, sitop, stats

### Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the top command.

### transaction

#### Description

The transaction command finds transactions based on events that meet various constraints. Transactions are made up of the raw text (the _raw field) of each member, the time and date fields of the earliest member, as well as the union of all other fields of each member.

Additionally, the transaction command adds two fields to the raw events, duration and eventcount. The values in the duration field show the difference between the timestamps for the first and last events in the transaction. The values in the eventcount field show the number of events in the transaction.

See About transactions in the Search Manual.
Syntax

transaction [<field-list>] [name=<transaction-name>] [txn_definition-options]... [memcontrol-options]... [rendering-options]...

Required arguments

None.

Optional arguments

field-list

Syntax: <field> ...
Description: One field or more field names. The events are grouped into transactions based on the values of this field. If a quoted list of fields is specified, events are grouped together if they have the same value for each of the fields.

memcontrol-options

Syntax: <maxopentxn> | <maxopenevents> | <keepevicted>
Description: These options control the memory usage for your transactions. They are not required, but you can use 0 or more of the options to define your transaction.

name

Syntax: name=<transaction-name>
Description: Specify the stanza name of a transaction that is configured in the transactiontypes.conf file. This runs the search using the settings defined in this stanza of the configuration file. If you provide other transaction definition options (such as maxspan) in this search, they overrule the settings in the configuration file.

rendering-options

Syntax: <delim> | <mvlist> | <mvraw> | <nullstr>
Description: These options control the multivalue rendering for your transactions. They are not required, but you can use 0 or more of the options to define your transaction.

txn_definition-options

Syntax: <maxspan> | <maxpause> | <maxevents> | <startwith> | <endswith> | <connected> | <unifyends> | <keeporphans>
Description: Specify the transaction definition options to define your transactions. You can use multiple options to define your transaction.
**Txn definition options**

**connected**

**Syntax:** connected=<bool>

**Description:** Only relevant if a field or fields list is specified. If an event contains fields required by the transaction, but none of these fields have been instantiated in the transaction (added with a previous event), this opens a new transaction (connected=true) or adds the event to the transaction (connected=false).

**Default:** true

**endswith**

**Syntax:** endswith=<filter-string>

**Description:** A search or eval expression which, if satisfied by an event, marks the end of a transaction.

**keeporphans**

**Syntax:** keeporphans=true | false

**Description:** Specify whether the transaction command should output the results that are not part of any transactions. The results that are passed through as "orphans" are distinguished from transaction events with a _txn_orphan field, which has a value of 1 for orphan results.

**Default:** false

**maxspan**

**Syntax:** maxspan=<int>[s | m | h | d]

**Description:** Specifies the maximum length of time in seconds, minutes, hours, or days that the events can span. The events in the transaction must span less than integer specified for maxspan. Events that exceed the maxspan limit are treated as part of a separate transaction. If the value is negative, the maxspan constraint is disabled and there is no limit.

**Default:** -1 (no limit)

**maxpause**

**Syntax:** maxpause=<int>[s | m | h | d]

**Description:** Specifies the maximum length of time in seconds, minutes, hours, or days for the pause between the events in a transaction. If value is negative, the maxpause constraint is disabled and there is no limit.

**Default:** -1 (no limit)

**maxevents**

**Syntax:** maxevents=<int>
**Description:** The maximum number of events in a transaction. If the value is negative this constraint is disabled.

**Default:** 1000

**startswith**

**Syntax:** startswith=<filter-string>

**Description:** A search or eval filtering expression which if satisfied by an event marks the beginning of a new transaction.

**unifyends**

**Syntax:** unifyends= true | false

**Description:** Whether to force events that match startswith/endswith constraint(s) to also match at least one of the fields used to unify events into a transaction.

**Default:** false

**Filter string options**

<filter-string>

**Syntax:** <search-expression> | (<quoted-search-expression>) | eval(<eval-expression>)

**Description:** A search or eval filtering expression which if satisfied by an event marks the end of a transaction.

<search-expression>

**Description:** A valid search expression that does not contain quotes.

<quote-search-expression>

**Description:** A valid search expression that contains quotes.

<eval-expression>

**Description:** A valid eval expression that evaluates to a Boolean.

**Memory constraint options**

If you have Splunk Cloud, Splunk Support administers the settings in the *limits.conf* file on your behalf.

**keepevicted**

**Syntax:** keepevicted=<bool>

**Description:** Whether to output evicted transactions. Evicted transactions can be distinguished from non-evicted transactions by checking the value of the 'closed_txn' field. The 'closed_txn' field is set to '0', or false, for
evicted transactions and '1', or true for non-evicted, or closed, transactions. The 'closed_txn' field is set to '1' if one of the following conditions is met: maxevents, maxpause, maxspan, startswith. For startswith, because the transaction command sees events in reverse time order, it closes a transaction when it satisfies the start condition. If none of these conditions is specified, all transactions are output even though all transactions will have 'closed_txn' set to '0'. A transaction can also be evicted when the memory limitations are reached.

**Default:** false or 0

maxopenevents

**Syntax:** maxopenevents=<int>

**Description:** Specifies the maximum number of events (which are) part of open transactions before transaction eviction starts happening, using LRU policy.

**Default:** The default value for this argument is read from the transactions stanza in the `limits.conf` file.

maxopentxn

**Syntax:** maxopentxn=<int>

**Description:** Specifies the maximum number of not yet closed transactions to keep in the open pool before starting to evict transactions, using LRU policy.

**Default:** The default value for this argument is read from the transactions stanza in the `limits.conf` file.

**Multivalue rendering options**

**delim**

**Syntax:** delim=<string>

**Description:** Specify a character to separate multiple values. When used in conjunction with mvraw=t, represents a string used to delimit the values of _raw.

**Default:** " " (whitespace)

**mvlist**

**Syntax:** mvlist= true | false | <field-list>

**Description:** Flag that controls how multivalued fields are processed. When set to mvlist=true, the multivalued fields in the transaction are a list of the original events ordered in arrival order. When set to mvlist=false, the multivalued fields in the transaction are a set of unique field values ordered alphabetically. If a comma or space delimited list of fields is provided, only those fields are rendered as lists.
**Default**: false

**mvraw**

**Syntax**: mvraw=<bool>

**Description**: Used to specify whether the `_raw` field of the transaction search result should be a multivalued field.

**Default**: false

**nullstr**

**Syntax**: nullstr=<string>

**Description**: A string value to use when rendering missing field values as part of multivalued fields in a transaction. This option applies only to fields that are rendered as lists.

**Default**: NULL

**Usage**

The `transaction` command is a centralized streaming command. See Command types.

If there are more than 5 events in a transaction, the remaining events are collapsed. A message appears at the end of the transaction which gives you the option to show all of the events in the transaction.

**Specifying multiple fields**

The Splunk software does not necessarily interpret the transaction defined by multiple fields as a conjunction (`field1 AND field2 AND field3`) or a disjunction (`field1 OR field2 OR field3`) of those fields. If there is a transitive relationship between the fields in the fields list and if the related events appear in the correct sequence, each with a different timestamp, `transaction` command will try to use it. For example, if you searched for

```
... | transaction host cookie
```

You might see the following events grouped into a transaction:

```
event=1 host=a
event=2 host=a cookie=b
event=3 cookie=b
```
**Descending time order required**

The `transaction` command requires that the incoming events be in descending time order. Some commands, such as `eval`, might change the order or time labeling of events. If one of these commands precedes the `transaction` command, your search returns an error unless you include a `sort` command in your search. The `sort` command must occur immediately before the `transaction` command to reorder the search results in descending time order.

**Basic Examples**

1. **Transactions with the same host, time range, and pause**

   Group search results that have the same host and cookie value, occur within 30 seconds, and do not have a pause of more than 5 seconds between the events.

   ```
   ... | transaction host cookie maxspan=30s maxpause=5s
   ```

2. **Transactions with the same "from" value, time range, and pause**

   Group search results that have the same value of "from", with a maximum span of 30 seconds, and a pause between events no greater than 5 seconds into a transaction.

   ```
   ... | transaction from maxspan=30s maxpause=5s
   ```

3. **Transactions with the same field values**

   You have events that include an alert_level. You want to create transactions where the level is equal. Using the `streamstats` command, you can remember the value of the alert level for the current and previous event. Using the `transaction` command, you can create a new transaction if the alert level is different. Output specific fields to table.

   ```
   ... | streamstats window=2 current=t latest(alert_level) AS last earliest(alert_level) AS first | transaction endswith=eval(first!=last) | table _time duration first last alert_level eventcount
   ```

**Extended Examples**
1. Transactions of Web access events based on IP address

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

Define a transaction based on Web access events that share the same IP address. The first and last events in the transaction should be no more than thirty seconds apart and each event should not be longer than five seconds apart.

```
sourcetype=access_* | transaction clientip maxspan=30s maxpause=5s
```

This produces the following events list. The clientip for each event in the transaction is highlighted.

This search groups events together based on the IP addresses accessing the server and the time constraints. The search results might have multiple values for some fields, such as host and source. For example, requests from a single IP could come from multiple hosts if multiple people are shopping from the same office. For more information, read the topic About transactions in the Knowledge Manager Manual.

2. Transaction of Web access events based on host and client IP

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions...
Define a transaction based on Web access events that have a unique combination of `host` and `clientip` values. The first and last events in the transaction should be no more than thirty seconds apart and each event should not be longer than five seconds apart.

```bash
sourcetype=access_* | transaction clientip host maxspan=30s maxpause=5s
```

This search produces the following events list.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
</table>

Each of these events have a distinct combination of the IP address (`clientip`) values and `host` values within the limits of the time constraints specified in the search.

### 3. Purchase transactions based on IP address and time range

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range `Yesterday` when you run the search.

This search defines a purchase transaction as 3 events from one IP address which occur in a 10 minute span of time.

```bash
sourcetype=access_* action=purchase | transaction clientip maxspan=10m maxevents=3
```

This search defines a purchase event based on Web access events that have the `action=purchase` value. These results are then piped into the `transaction`
command. This search identifies purchase transactions by events that share the same `clientip`, where each session lasts no longer than 10 minutes, and includes no more than 3 events.

This search produces the following events list:

```
sourcetype="cisco:esa" | transaction mid dcid icid maxevents=10 endswith="Message done"
```

This search produces the following list of events:

4. Email transactions based on maxevents and endswith

This example uses sample email data. You should be able to run this search on any email data by replacing the `sourcetype=cisco:esa` with the `sourcetype` value and the `mailfrom` field with email address field name in your data. For example, the email might be To, From, or Cc).

This example defines an email transaction as a group of up to 10 events. Each event contains the same value for the mid (message ID), icid (incoming connection ID), and dcid (delivery connection ID). The last event in the transaction contains a Message done string.

```
sourcetype="cisco:esa" | transaction mid dcid icid maxevents=10 endswith="Message done"
```

This search produces the following list of events:
By default, only the first 5 events in a transaction are shown. The first transaction contains 7 events and the last event is hidden. The second and third transactions show the **Message done** string in the last event in the transaction.

5. **Email transactions based on maxevents, maxspan, and mvlist**

This example uses sample email data. You should be able to run this search on any email data by replacing the `sourcetype=cisco:esa` with the `sourcetype` value and the `mailfrom` field with email address field name in your data. For example, the email might be To, From, or Cc).

This example defines an email transaction as a group of up to 10 events. Each event contains the same value for the `mid` (message ID), `icid` (incoming connection ID), and `dcid` (delivery connection ID). The first and last events in the transaction should be no more than thirty seconds apart.

```
sourcetype="cisco:esa" | transaction mid dcid icid maxevents=10 maxspan=30s mvlist=true
```

By default, the values of multivalue fields are suppressed in search results with the default setting for `mvlist`, which is false. Specifying `mvlist=true` in this search displays all of the values of the selected fields. This produces the following events list:
Here you can see that each transaction has a duration that is less than thirty seconds. Also, if there is more than one value for a field, each of the values is listed.

6. Transactions with the same session ID and IP address

This example uses the sample data from the Search Tutorial but should work with any format of Apache web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

Define a transaction as a group of events that have the same session ID, JSESSIONID, and come from the same IP address, clientip, and where the first event contains the string, "view", and the last event contains the string, "purchase".

```
sourcetype=access_* | transaction JSESSIONID clientip startswith="view" endswith="purchase" | where duration>0
```

The search defines the first event in the transaction as events that include the string, "view", using the `startswith="view"` argument. The `endswith="purchase"` argument does the same for the last event in the transaction.

This example then pipes the transactions into the `where` command and the `duration` field to filter out all of the transactions that took less than a second to complete. The `where` filter cannot be applied before the `transaction` command because the `duration` field is added by the `transaction` command.
You might be curious about why the transactions took a long time, so viewing these events might help you to troubleshoot.

You won’t see it in this data, but some transactions might take a long time because the user is updating and removing items from their shopping cart before they completes the purchase. Additionally, this search is run over all events. There is no filtering before the `transaction` command. Anytime you can filter the search before the first pipe, the faster the search runs.

### See also

Reference

-About transactions

Commands

- `stats`, `concurrency`

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the transaction command.

### transpose

**Description**

Returns the specified number of rows (search results) as columns (list of field values), such that each search row becomes a column.
Syntax

transpose [int] [column_name=<string>] [header_field=<field>] [include_empty=<bool>]

**Required arguments**

None.

**Optional arguments**

column_name

Synt**ax**: column_name=<string>

**Description**: The name of the first column that you want to use for the transposed rows. This column contains the names of the fields.

**Default**: column

header_field

**Syntax**: header_field=<field>

**Description**: The field in your results to use for the names of the columns (other than the first column) in the transposed data.

**Default**: row 1, row 2, row 3, and so on.

include_empty

**Syntax**: include_empty=<bool>

**Description**: Specify whether to include (true) or not include (false) fields that contain empty values.

**Default**: true

int

**Syntax**: <int>

**Description**: Limit the number of rows to transpose. To transpose all rows, specify `transpose 0`, which indicates that the number of rows to transpose is unlimited.

**Default**: 5

**Usage**

When you use the `transpose` command the field names used in the output are based on the arguments that you use with the command. By default the field names are: `column, row1, row2, and so forth.`
Examples

1. **Transpose the results of a chart command**

Use the default settings for the transpose command to transpose the results of a chart command.

```
... | chart count BY host error_code | transpose
```

2. **Count the number of events by sourcetype and transpose the results to display the 3 highest counts**

Count the number of events by sourcetype and display the sourcetypes with the highest count first.

```
index=_internal | stats count by sourcetype | sort -count
```

![Table with sourcetypes and counts]

Use the transpose command to convert the rows to columns and show the source types with the 3 highest counts.

```
index=_internal | stats count by sourcetype | sort -count | transpose 3
```

![Table with transposed sourcetypes and counts]
3. Transpose a set of data into a series to produce a chart

This example uses the sample dataset from the Search Tutorial.

- Download the data set from Add data tutorial and follow the instructions to get the tutorial data into your Splunk deployment.

Search all successful events and count the number of views, the number of times items were added to the cart, and the number of purchases.

```
sourcetype=access_* status=200 | stats count AS views
count(eval(action="addtocart")) AS addtocart
count(eval(action="purchase")) AS purchases
```

This search produces a single row of data.

The value for `count AS views` is the total number of the events that match the criteria `sourcetype=access_* status=200`, or the total count for all actions. The values for `addtocart` and `purchases` show the number of events for those specific actions.

When you switch to the Visualization tab, the data displays a chart with the "34282 views" as the X axis label and two columns, one for "addtocart" and one for "purchases". Because the information about the views is placed on the X axis, this chart is confusing.
If you change to a pie chart, you see only the "views".
Use the `transpose` command to convert the columns of the single row into multiple rows.

```
sourcetype=access_* status=200 | stats count AS views
     count(eval(action="addtocart")) AS addtocart
     count(eval(action="purchase")) AS purchases | transpose
```

Now these rows can be displayed in a column or pie chart where you can compare the values.
In this particular example, using a pie chart is misleading. The views is a total count of all the actions, not just the addtocart and purchases actions. Using a pie chart implies that views is an action like addtocart and purchases. The pie chart implies that the value for views is 1 part of the total, when in fact views is the total.

There are a few ways to fix this issue:

- Use a column chart
- You can remove the count AS views criteria from your search
- You can add the table command before the transpose command in the search, for example:

```plaintext
sourcetype=access_* status=200 | stats count AS views count(eval(action="addtocart")) AS addtocart count(eval(action="purchase")) AS purchases | table addtocart purchases | transpose
```

See also

fields, stats

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the transpose command.

trendline

Description

Computes the moving averages of fields: simple moving average (sma), exponential moving average (ema), and weighted moving average (wma) The output is written to a new field, which you can specify.

SMA and WMA both compute a sum over the period of most recent values. WMA puts more weight on recent values rather than past values. EMA is calculated using the following formula.

\[ EMA(t) = \alpha \times EMA(t-1) + (1 - \alpha) \times field(t) \]

where \( \alpha = 2/(period + 1) \) and \( field(t) \) is the current value of a field.
Syntax

trendline ( <trendtype><period>"(<field>)" [AS <newfield>] ) ... 

Required arguments

trendtype

Syntax: syntax = sma | ema | wma
Description: The type of trend to compute. Current supported trend types include simple moving average (sma), exponential moving average (ema), and weighted moving average (wma).

period

Syntax: <num>
Description: The period over which to compute the trend, an integer between 2 and 10000.

<field>

Syntax: "(<field>)"
Description: The name of the field on which to calculate the trend.

Optional arguments

<newfield>

Syntax: <field>
Description: Specify a new field name to write the output to.
Default: <trendtype><period>(<field>)

Usage

Examples

Example 1: Computes a five event simple moving average for field 'foo' and writes the result to new field called 'smoothed_foo.' Also, in the same line, computes ten event exponential moving average for field 'bar'. Because no AS clause is specified, writes the result to the field 'ema10(bar).

... | trendline sma5(foo) AS smoothed_foo ema10(bar)

Example 2: Overlay a trendline over a chart of events by month.

index="bar" | stats count BY date_month | trendline sma2(count) AS trend | fields * trend

671
See also

accum, autoregress, delta, streamstats

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the trendline command.

tscollect

<table>
<thead>
<tr>
<th>This feature is deprecated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <code>tscollect</code> command is deprecated in the Splunk platform as of version 7.3.0. Although this command continues to function, it might be removed in a future version. This command has been superseded by data models. See Accelerate data models in the Knowledge Manager Manual.</td>
</tr>
</tbody>
</table>

In the version 7.3.0 Release Notes, see Deprecated features.

Description

The `tscollect` command uses indexed fields to create **time series index (tsidx) files** in a namespace that you define. The result tables in these files are a subset of the data that you have already indexed. This then enables you to use the `tstats` command to search and report on these tsidx files instead of searching raw data. Because you are searching on a subset of the full index, the search should complete faster than it would otherwise.

The `tscollect` command creates multiple tsidx files in the same namespace. The command will begin a new tsidx file when it determines that the tsidx file it is currently creating has gotten big enough.

Only users with the `indexes_edit` capability can run this command. See Usage.

Syntax

```
... | tscollect [namespace=<string>] [squashcase=<bool>] [keepresults=<bool>]
```
Optional arguments

keepresults

Syntax: keepresults = true | false
Description: If true, tscollect outputs the same results it received as input. If false, tscollect returns the count of results processed (this is more efficient since it does not need to store as many results).
Default: false

namespace

Syntax: namespace=<string>
Description: Define a location for the tsidx file(s). If namespace is provided, the tsidx files are written to a directory of that name under the main tsidxstats directory (that is, within $SPLUNK_DB/tsidxstats). These namespaces can be written to multiple times to add new data.
Default: If namespace is not provided, the files are written to a directory within the job directory of that search, and will live as long as the job does. If you have Splunk Enterprise, you can configure the namespace location by editing indexes.conf and setting the attribute tsidxStatsHomePath.

squashcase

Syntax: squashcase = true | false
Description: Specify whether or not the case for the entire field::value tokens are case sensitive when it is put into the lexicon. To create indexed field tsidx files that are similar to those created by Splunk Enterprise, set squashcase=true for results to be converted to all lowercase.
Default: false

Usage

You must have the indexes_edit capability to run the tscollect command. By default, the admin role has this capability and the user and power roles do not have this capability.

Examples

Example 1: Write the results table to tsidx files in namespace foo.

... | tscollect namespace=foo

Example 2: Retrieve events from the main index and write the values of field foo to tsidx files in the job directory.
See also

collect, stats, tstats

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the tscollect command.

tstats

Description

Use the tstats command to perform statistical queries on indexed fields in tsidx files. The indexed fields can be from normal index data, tscollect data, or accelerated data models.

Syntax

| tstats [prestats=<bool>] [local=<bool>] [append=<bool>] [summariesonly=<bool>] [allow_old_summaries=<bool>] [chunk_size=<unsigned int>] <stats-func>... [FROM ( <namespace> | sid=<tscollect-job-id> | datamodel=<data_model-name> )] [WHERE <search-query> | <field> IN (<value-list>)] [BY <field-list> [span=<timespan>]]

Required arguments

<stats-func>

Syntax: (count [<field>] | <function>(<field>))... [AS <string>]

Description: Either perform a basic count of a field or perform a function on a field. For a list of the supported functions for the tstats command, refer to the table below. You can specify one or more functions. You can also rename the result using the AS keyword, unless you are in prestats mode. You cannot use wildcards to specify field names. See Usage.

The following table lists the supported functions by type of function. Use the links in the table to see descriptions and examples for each function.
For an overview about using functions with commands, see Statistical and charting functions.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate functions</td>
<td>avg()</td>
</tr>
<tr>
<td></td>
<td>count()</td>
</tr>
<tr>
<td></td>
<td>distinct_count()</td>
</tr>
<tr>
<td></td>
<td>estdc()</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Event order functions</td>
<td>first()</td>
</tr>
<tr>
<td>Multivalue stats and chart functions</td>
<td>values</td>
</tr>
<tr>
<td>Time functions</td>
<td>earliest()</td>
</tr>
<tr>
<td></td>
<td>earliest_time()</td>
</tr>
</tbody>
</table>

Optional arguments

append

Syntax: append=<bool>
Description: When in prestats mode (prestats=t), enables append=t where the prestats results append to existing results, instead of generating them.
Default: false

allow_old_summaries

Syntax: allow_old_summaries=true | false
Description: Only applies when selecting from an accelerated data model. To return results from summary directories only when those directories are up-to-date, set this parameter to false. If the data model definition has changed, summary directories that are older than the new definition are not used when producing output from tstats. This default ensures that the output from tstats will always reflect your current configuration. When set to true, tstats will use both current summary data |
and summary data that was generated prior to the definition change. Essentially this is an advanced performance feature for cases where you know that the old summaries are "good enough".

**Default:** false

**chunk_size**

**Syntax:** chunk_size=<unsigned_int>

**Description:** Advanced option. This argument controls how many events are retrieved at a time within a single TSIDX file when answering queries. Only consider supplying a lower value for this if you find a particular query is using too much memory. The case that could cause this would be an excessively high cardinality split-by, such as grouping by several fields that have a very large amount of distinct values. Setting this value too low can negatively impact the overall run time of your query.

**Default:** 10000000 (10 million)

**local**

**Syntax:** local=true | false

**Description:** If true, forces the processor to be run only on the search head.

**Default:** false

**prestats**

**Syntax:** prestats=true | false

**Description:** Specifies whether to use the prestats format. The prestats format is a Splunk internal format that is designed to be consumed by commands that generate aggregate calculations. When using the prestats format you can pipe the data into the `chart`, `stats`, or `timechart` commands, which are designed to accept the prestats format. When `prestats=true`, AS instructions are not relevant. The field names for the aggregates are determined by the command that consumes the prestats format and produces the aggregate output.

**Default:** false

**summariesonly**

**Syntax:** summariesonly=<bool>

**Description:** Only applies when selecting from an accelerated data model. When false, generates results from both summarized data and data that is not summarized. For data not summarized as TSIDX data, the full search behavior will be used against the original index data. If set to true, 'tstats' will only generate results from the TSIDX data that has been automatically generated by the acceleration and non-summarized data will not be provided.
Default: false

FROM clause arguments

The FROM clause is optional. You can specify either a namespace, an sid, or a datamodel. See Selecting data for more information about this clause.

namespace
  Syntax: <string>
  Description: Define a location for the tsidx file with $SPLUNK_DB/tsidxstats. If you have Splunk Enterprise, you can configure this location by editing the local version of the indexes.conf file and setting the tsidxStatsHomePath attribute. See How to edit a configuration file in the Admin manual.

sid
  Syntax: sid=<tscollect-job-id>
  Description: The job ID string of a tscollect search (that generated tsidx files).

datamodel
  Syntax: datamodel=<data_model-name>
  Description: The name of an accelerated data model.

WHERE clause arguments

The WHERE clause is optional. This clause is used as a filter. You can specify either a search or a field and a set of values with the IN operator.

<search-query>
  Specify search criteria to filter on.

<field> IN (<value-list>)
  For the field, specify a list of values to include in the search results.

BY clause arguments

The BY clause is optional. You cannot use wildcards in the BY clause with the tstats command. See Usage. If you use the BY clause, you must specify a field-list. You can also specify a span.

<field-list>
  Syntax: <field>, ...

677
**Description:** Specify one or more fields to group results.

**span**

**Syntax:** `span=<timespan>`

**Description:** The span of each time bin. If you use the BY clause to group by `_time`, use the `span` argument to group the time buckets. You can specify timespans such as `BY _time span=1h` or `BY _time span=5d`. If you do not specify a `<timespan>`, the default is `auto`, which means that the number of time buckets adjusts to produce a reasonable number of results. For example if initially seconds are used for the `<timespan>` and too many results are being returned, the `<timespan>` is changed to a longer value, such as minutes, to return fewer time buckets.

**Default:** auto

**<timespan>**

**Syntax:** `auto | <int><timescale>`

**<timescale>**

**Syntax:** `<sec> | <min> | <hr> | <day> | <month>`

**Description:** Time scale units. For the `tstats` command, the `<timescale>` does not support subseconds.

**Default:** `sec`

<table>
<thead>
<tr>
<th>Time scale</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;sec&gt;</code></td>
<td>`s</td>
<td>sec</td>
</tr>
<tr>
<td><code>&lt;min&gt;</code></td>
<td>`m</td>
<td>min</td>
</tr>
<tr>
<td><code>&lt;hr&gt;</code></td>
<td>`h</td>
<td>hr</td>
</tr>
<tr>
<td><code>&lt;day&gt;</code></td>
<td>`d</td>
<td>day</td>
</tr>
<tr>
<td><code>&lt;month&gt;</code></td>
<td>`mon</td>
<td>month</td>
</tr>
</tbody>
</table>

**Usage**

The `tstats` command is a report-generating command, except when `prestats=true`. When `prestats=true`, the `tstats` command is an event-generating command. See Command types.
Generating commands use a leading pipe character and should be the first command in a search, except when `prestats=true`.

**Wildcard characters**

The `tstats` command does not support wildcard characters in field values in aggregate functions or BY clauses.

For example, you cannot specify `| tstats avg(foo*)` or `| tstats count WHERE host=x BY source*`.

Samples of aggregate functions include `avg()`, `count()`, `max()`, `min()`, and `sum()`.

Any results returned where the aggregate function or BY clause includes a wildcard character are only the most recent few minutes of data that has not been summarized. Include the `summariesonly=t` argument with your `tstats` command to return only summarized data.

**Nested eval expressions not supported**

You cannot use `eval` expressions inside aggregate functions with the `tstats` command.

For example, `| tstats count(eval(...))` is not supported.

While nested eval expressions are supported with the `stats` command, they are not supported with the `tstats` command.

**Functions and memory usage**

Some functions are inherently more expensive, from a memory standpoint, than other functions. For example, the `distinct_count` function requires far more memory than the `count` function. The `values` and `list` functions also can consume a lot of memory.

If you are using the `distinct_count` function without a split-by field or with a low-cardinality split-by field, consider replacing the `distinct_count` function with the the `estdc` function (estimated distinct count). The `estdc` function might result in significantly lower memory usage and run times.
**Memory and maximum results**

In the `limits.conf` file, the `maxresultrows` setting in the `[searchresults]` stanza specifies the maximum number of results to return. The default value is 50,000. Increasing this limit can result in more memory usage.

The `max_mem_usage_mb` setting in the `[default]` stanza is used to limit how much memory the `tstats` command uses to keep track of information. If the `tstats` command reaches this limit, the command stops adding the requested fields to the search results. You can increase the limit, contingent on the available system memory.

If you are using Splunk Cloud and want to change either of these limits, file a Support ticket.

**Complex aggregate functions**

The `tstats` command does not support complex aggregate functions such as

```
...count(eval('Authentication.action'=='failure')).
```

Consider the following query. This query will not return accurate results because complex aggregate functions are not supported by the `tstats` command.

```
| tstats summariesonly=false values(Authentication.tag) as tag,
values(Authentication.app) as app,
count(eval('Authentication.action'=='failure')) as failure,
count(eval('Authentication.action'=='success')) as success from
datamodel=Authentication by Authentication.src | search success>0 | where failure > 5 | `settags("access")` | `drop_dm_object_name("Authentication")` |
```

Instead, separate out the aggregate functions from the eval functions, as shown in the following search.

```
| tstats `summariesonly` values(Authentication.app) as app, count from
datamodel=Authentication by Authentication.action,
Authentication.src | `drop_dm_object_name("Authentication")` | eval
success=if(action="success",count,0),
failure=if(action="failure",count,0) | stats values(app) as app,
sum(failure) as failure, sum(success) as success by src
```

**Sparkline charts**

You can generate sparkline charts with the `tstats` command only if you specify the `_time` field in the `BY` clause and use the `stats` command to generate the
actual sparkline. For example:

```
| tstats count from datamodel=Authentication.Authentication BY _time, Authentication.src span=1h | stats sparkline(sum(count),1h) AS sparkline, sum(count) AS count BY Authentication.src
```

**Selecting data**

Use the `tstats` command to perform statistical queries on indexed fields in `.tsidx` files. You can select the data for the indexed fields in several ways.

**Normal index data**

Use a `FROM` clause to specify a namespace, search job ID, or data model. If you do not specify a `FROM` clause, the Splunk software selects from index data in the same way as the `search` command. You are restricted to selecting data from your allowed indexes by user role. You control exactly which indexes you select data from by using the `WHERE` clause. If no indexes are mentioned in the `WHERE` clause, the Splunk software uses the default indexes. By default, role-based search filters are applied, but can be turned off in the `limits.conf` file.

**Data manually collected with the tscollect command**

You can select data from your namespace by specifying `FROM <namespace>`. If you did not specify a namespace with the `tscollect` command, the data is collected into the dispatch directory of that job. If the data is in the dispatch directory, you select the data by specifying `FROM sid=<tscollect-job-id>`.

**An accelerated data model**

You can select data from a high-performance analytics store, which is a collection of `.tsidx` data summaries, for an accelerated data model. You can select data from this accelerated data model by using `FROM datamodel=<data_model_name>`.

Search filters cannot be applied to accelerated data models. This includes both role-based and user-based search filters.

**An accelerated data model dataset**

When you select data within an accelerated data model, you can further constrain your search by indicating a dataset within that data model that you want to select data from. You do this by using a `WHERE` clause to indicate the `nodename` of the data model dataset. The `nodename` value indicates where the dataset is in a data model hierarchy.
When you use `nodename` in a search, you always use the following construction:

```
FROM datamodel=<data_model_name> where
nodename=<root_dataset_name>.<parent_dataset_name>.<...>.<target_dataset_name>
```

For example, suppose you want to search on a dataset named `scheduled_reports` in your `internal_server` data model. In that data model, the `scheduled_reports` dataset is a child of the `scheduler` dataset, which in turn is a child of the `server` root event dataset. This means that you should represent the `scheduled_report` dataset in your search as `nodename=server.scheduler.scheduled_reports`.

If you run that search and decide you want to search on the contents of the `scheduler` data model dataset instead, you would use `nodename=server.scheduler` in your new search.

Search filters cannot be applied to accelerated data model datasets. This includes both role-based and user-based search filters.

You might see a count mismatch in the events retrieved when searching `tsidx` files. It is not possible to distinguish between indexed field tokens and raw tokens in `tsidx` files. On the other hand, it is more explicit to run the `tstats` command on accelerated data models or from a `tscollect` command, where only the fields and values are stored and not the raw tokens.

**Filtering with WHERE**

You can provide any number of aggregates (`aggregate-opt`) to perform and also have the option of providing a filtering query using the `WHERE` keyword. This query looks like a normal query you would use in the search processor. This supports all the same time arguments as search, such as `earliest=-1y`.

**Grouping by _time**

You can provide any number of `GROUPBY` fields. If you are grouping by `_time`, supply a timespan with `span` for grouping the time buckets, for example `...BY _time span=1h` or `...BY _time span=3d`.

**Examples**

**Example 1:** Gets the count of all events in the `mydata` namespace.

```
| tstats count FROM mydata
```

682
Example 2: Returns the average of the field foo in mydata, specifically where bar is value2 and the value of baz is greater than 5.

| tstats avg(foo) FROM mydata WHERE bar=value2 baz>5

Example 3: Gives the count by source for events with host=x.

| tstats count WHERE host=x BY source

Example 4: Gives a timechart of all the data in your default indexes with a day granularity.

| tstats prestats=t count BY _time span=1d | timechart span=1d count

Example 5: Use prestats mode in conjunction with append to compute the median values of foo and bar, which are in different namespaces.

| tstats prestats=t median(foo) FROM mydata | tstats prestats=t append=t median(bar) FROM otherdata | stats median(foo) median(bar)

Example 6: Uses the summariesonly argument to get the time range of the summary for an accelerated data model named mydm.

| tstats summariesonly=t min(_time) AS min, max(_time) AS max FROM datamodel=mydm | eval prettymin=strftime(min, "%c") | eval prettymax=strftime(max, "%c")

Example 7: Uses summariesonly in conjunction with timechart to reveal what data has been summarized over the past hour for an accelerated data model titled mydm.

| tstats summariesonly=t prestats=t count FROM datamodel=mydm BY _time span=1h | timechart span=1h count

Example 8: Uses the values statistical function to provide lists of values for each field returned by the "Splunk's Internal Server Logs" data model.

| tstats values FROM datamodel=internal_server.server

Example 9: Uses the values statistical function to provide lists of values for each field returned by the Alerts dataset within the "Splunk's Internal Server Logs" data model.

| tstats values FROM datamodel=internal_server where nodename=server.scheduler.alerts
See also

stats, tscollect

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the tstats command.

typeahead

Description

Returns typeahead information for a specified prefix. The maximum number of results returned is based on value you specify for the `count` argument. The `typeahead` command can be targeted to an index and restricted by time.

Syntax

```
| typeahead prefix=<string> count=<int> [max_time=<int>] [index=<string>] [starttimeu=<int>] [endtimeu=<int>] [collapse=<bool>]
```

**Required arguments**

prefix

**Syntax:** prefix=<string>

**Description:** The full search string to return typeahead information.

count

**Syntax:** count=<int>

**Description:** The maximum number of results to return.

**Optional arguments**

index-specifier

**Syntax:** index=<string>

**Description:** Search the specified index instead of the default index.

max_time

**Syntax:** max_time=<int>
Description: The maximum time in seconds that the typeahead can run. If max_time=0, there is no limit.

starttimeu
Syntax: starttimeu=<int>
Description: Set the start time to N seconds, measured in UNIX time.
Default: 0

detimeu
Syntax: endtimeu=<int>
Description: Set the end time to N seconds, measured in UNIX time.
Default: now

collapse
Syntax: collapse=<bool>
Description: Specify whether to collapse a term that is a prefix of another term when the event count is the same.
Default: true

Typeahead and sourcetype renaming

After renaming the sourcetype in the props.conf file, it takes about 5 minutes (the exact time might slightly depend on the performance of the server) to clear up the cache data. A typeahead search that is run while the cache is being cleared returns the cached source type data. This is expected behavior.

To remove the cached data, in a terminal window run the following command:

```
rm $SPLUNK_HOME/var/run/splunk/typeahead/*
```

When you re-run the typeahead search, you should see the renamed source types.

For more information, see Rename source types in the Getting Data In manual.

Usage

The typeahead command is a generating command and should be the first command in the search. Generating commands use a leading pipe character.
Examples

**Example 1:**

Return typeahead information for sources in the "_internal" index.

```
| typeahead prefix=source count=10 index=_internal
```

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the typeahead command.

**typelearner**

The *typelearner* command is deprecated. Use the *findtypes* command instead.

**Description**

Generates suggested event types by taking previous search results and producing a list of potential searches that can be used as event types. By default, the *typelearner* command initially groups events by the value of the grouping-field. The search then unifies and merges these groups based on the keywords they contain.

**Syntax**

```
typelearner [ <grouping-field> ] [ <grouping-maxlen> ]
```

**Optional arguments**

**grouping-field**

- **Syntax:** `<field>`
- **Description:** The field with values for the *typelearner* command to use when initially grouping events.
- **Default:** punct, the punctuation seen in _raw_

**grouping-maxlen**

- **Syntax:** `maxlen=<int>`
- **Description:** Determines how many characters in the grouping-field value to look at. If set to negative, the entire value of the grouping-field value is...
used to group events.

Default: 15

Examples

Example 1:

Have the search automatically discover and apply event types to search results.

... | typelearner

See also

typer

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the typelearner command.

typer

Description

Calculates the 'eventtype' field for search results that match a known event type.

Syntax

typer

Usage

The typer command is a distributable streaming command. See Command types.

By default, the typer command looks at the first 10000 characters of an event to determine the event type. You can override this default in your search by specifying maxlen=<int> with the typer command. For example:

... | typer maxlen=300
**Changing the default for maxlen**

Users with file system access, such as system administrators, can change the default setting for maxlen.

**Prerequisite**

- Review the steps in How to edit a configuration file in the *Admin Manual*.

Never change or copy the configuration files in the default directory. The files in the default directory must remain intact and in their original location. Make the changes in the local directory.

**Steps**

1. Open the local `limits.conf` file for the Search app. For example, $SPLUNK_HOME/etc/apps/<app_name>/local.
2. Under the [typer] stanza, specify the default for the maxlen setting.

If you are using Splunk Cloud and want to change the default, open a Support ticket.

**Examples**

**Example 1:**

Force the search to apply event types that you have configured (Splunk Web automatically does this when you view the "eventtype" field).

```plaintext
... | typer
```

**See also**

typelearner

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the typer command.
union

Description

Merges the results from two or more datasets into one dataset. One of the datasets can be a result set that is then piped into the `union` command and merged with a second dataset.

The `union` command appends or merges event from the specified datasets, depending on whether the dataset is streaming or non-streaming and where the command is run. The `union` command runs on indexers in parallel where possible, and automatically interleaves results on the `_time` when processing events. See Usage.

If you are familiar with SQL but new to SPL, see Splunk SPL for SQL users.

Syntax

```
union [subsearch-options] <dataset> [dataset>...]
```

Required arguments

dataset

Syntax: `<dataset-type>:<dataset-name> | <subsearch>`
Description: The dataset that you want to perform the union on. The dataset can be either a named or unnamed dataset.
◊ A named dataset is comprised of `<dataset-type>:<dataset-name>`. For `<dataset-type>` you can specify a data model, a saved search, or an inputlookup. For example `datamodel:"internal_server.splunkdaccess"`.
◊ A subsearch is an unnamed dataset.

When specifying more than one dataset, use a space or a comma separator between the dataset names.

Optional arguments

subsearch-options

Syntax: `maxtime=<int> maxout=<int> timeout=<int>`
Description: You can specify one set of subsearch-options that apply to all of the subsearches. You can specify one or more of the subsearch-options. These options apply only when the subsearch is
treated as a non-streaming search.
◊ The `maxtime` argument specifies the maximum number of seconds to run the subsearch before finalizing. The default is 60 seconds.
◊ The `maxout` argument specifies the maximum number of results to return from the subsearch. The default is 50000 results. This value is the `maxresultrows` setting is in the [searchresults] stanza in the `limits.conf` file.
◊ The `timeout` argument specifies the maximum amount of time, in seconds, to cache the subsearch results. The default is 300 seconds.

**Usage**

The `union` command is a generating command.

How the `union` command processes datasets depends on whether the dataset is a streaming or non-streaming dataset. The type of dataset is determined by the commands that are used to create the dataset. See Types of commands.

There are two types of streaming commands, distributable streaming and centralized streaming. For this discussion about the `union` command, streaming datasets refers to distributable streaming.

*Where the command is run*

Whether the datasets are streaming or non-streaming determines if the `union` command is run on the indexers or the search head. The following table specifies where the command is run.

<table>
<thead>
<tr>
<th>Dataset type</th>
<th>Dataset 1 is streaming</th>
<th>Dataset 1 is non-streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 2 is streaming</td>
<td>Indexers</td>
<td>Search head</td>
</tr>
<tr>
<td>Dataset 2 is non-streaming</td>
<td>Search head</td>
<td>Search head</td>
</tr>
</tbody>
</table>

*How the command is processed*

The type of dataset also determines how the `union` command is processed.

<table>
<thead>
<tr>
<th>Dataset type</th>
<th>Impact on processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized streaming or non-streaming</td>
<td>Processed as an <code>append</code> command.</td>
</tr>
<tr>
<td>Dataset type</td>
<td>Impact on processing</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Distributable streaming</td>
<td>Processed as a <strong>multisearch</strong> command.</td>
</tr>
<tr>
<td></td>
<td>Placing <code>&lt;streaming_dataset1&gt;</code> after the <code>union</code> command is more efficient.</td>
</tr>
</tbody>
</table>

**Optimized syntax for streaming datasets**

With streaming datasets, instead of this syntax:

```plaintext
<streaming_dataset1> | union <streaming_dataset2>
```

Your search is more efficient with this syntax:

```plaintext
... | union <streaming_dataset1>, <streaming_dataset2>
```

**Why unioned results might be truncated**

Consider the following search, which uses the `union` command to merge the events from three indexes. Each index contains 60,000 events, for a total of 180,000 events.

```plaintext
| union maxout=10000000 [ search index=union_1 ] [ search index=union_2 ] [ search index=union_3 ] | stats count by index
```

This search produces the following union results:

<table>
<thead>
<tr>
<th>index</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>union_1</td>
<td>60000</td>
</tr>
<tr>
<td>union_2</td>
<td>60000</td>
</tr>
<tr>
<td>union_3</td>
<td>60000</td>
</tr>
</tbody>
</table>

In this example, all of the subsearches are distributable streaming, so they are unioned by using same processing as the **multisearch** command. All 60,000 results for each index are unioned for a total of 180,000 merged events.

However, if you specify a centralized streaming command, such as the **head** command, in one of the subsearches the results change.
This search produces the following union results for a total of 160,000 merged events.

<table>
<thead>
<tr>
<th>index</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>union_1</td>
<td>60000</td>
</tr>
<tr>
<td>union_2</td>
<td>50000</td>
</tr>
<tr>
<td>union_3</td>
<td>50000</td>
</tr>
</tbody>
</table>

Because the `head` command is a centralized streaming command rather than distributable streaming command, any subsearches that follow the `head` command are processed using the `append` command. In other words, when a command forces the processing to the search head, all subsequent commands must also be processed on the search head.

Internally, the search is converted to this:

```
| search index=union_1 | head 60000 | append [ search index=union_2 ] | append [ search index=union_3 ] | stats count by index
```

When the `union` command is used with commands that are non-streaming commands, the default for the `maxout` argument is enforced. The default for the `maxout` argument is 50,000 events. In this example, the default for the `maxout` argument is enforced starting with the subsearch that used the non-streaming command. The default is enforced for any subsequent subsearches.

If the non-streaming command is on the last subsearch, the first two subsearches are processed as streaming. These subsearches are unioned using the `multisearch` command processing. The final subsearch includes a non-streaming command, the `head` command. That subsearch gets unioned using the `append` command processing.

Internally this search is converted to this:

```
| multisearch [ search index=union_1 ] [ search index=union_2 ] | | append [ search index=union_3 | head 60000 ] | stats count by index
```

In this example, the default for the `maxout` argument applies only to the last subsearch. That subsearch returns only 50,000 events instead of the entire set of 60,000 events. The total number events merged is 170,000. 60,000 events for the first and second subsearches and 50,000 events from the last subsearch.
**Interleaving results**

When two datasets are retrieved from disk in time descending order, which is the default sort order, the `union` command interleaves the results. The interleave is based on the `_time` field. For example, you have the following datasets:

**dataset_A**

<table>
<thead>
<tr>
<th>_time</th>
<th>host</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>mailsrv1</td>
<td>2412</td>
</tr>
<tr>
<td>1</td>
<td>dns15</td>
<td>231</td>
</tr>
</tbody>
</table>

**dataset_B**

<table>
<thead>
<tr>
<th>_time</th>
<th>host</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>router1</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>dns12</td>
<td>22o</td>
</tr>
</tbody>
</table>

Both datasets are descending order by `_time`. When `| union dataset_A, dataset_B` is run, the following dataset is the result.

<table>
<thead>
<tr>
<th>_time</th>
<th>host</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>mailsrv1</td>
<td>2412</td>
</tr>
<tr>
<td>3</td>
<td>router1</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>dns12</td>
<td>22o</td>
</tr>
<tr>
<td>1</td>
<td>dns15</td>
<td>231</td>
</tr>
</tbody>
</table>

**Examples**

1. **Union events from two subsearches**

   The following example merges events from `index a` and `index b`. New fields `type` and `mytype` are added in each subsearch using the `eval` command.

   ```sh
   | union [search index=a | eval type = "foo"] [search index=b | eval mytype = "bar"]
   ```

2. **Union the results of a subsearch to the results of the main search**

   The following example appends the current results of the main search with the tabular results of errors from the subsearch.
3. **Union events from a data model and events from an index**

The following example unions a built-in data model that is an internal server log for REST API calls and the events from index `a`.

```plaintext
... | union datamodel:"internal_server.splunkdaccess" [search index=a]
```

4. **Specify the subsearch options**

The following example sets a maximum of 20,000 results to return from the subsearch. The example specifies to limit the duration of the subsearch to 120 seconds. The example also sets a maximum time of 600 seconds (5 minutes) to cache the subsearch results.

```plaintext
... | chart count by category1 | union maxout=20000 maxtime=120 timeout=600 [search error | chart count by category2]
```

**See also**

- About subsearches in the *Search Manual*
- About data models in the *Knowledge Manager Manual*
- `search`, `inputlookup`

**uniq**

**Description**

The `uniq` command works as a filter on the search results that you pass into it. This command removes any search result if that result is an exact duplicate of the previous result. This command does not take any arguments.

**Note:** We do not recommend running this command against a large dataset.

**Syntax**

```plaintext
uniq
```
Examples

Example 1:

Keep only unique results from all web traffic in the past hour.

```
eventtype=webtraffic earliest=-1h@s | uniq
```

See also
dedup

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the uniq command.

untable

Description

Converts results from a tabular format to a format similar to stats output. This command is the inverse of the xyseries command.

Syntax

```
untable <x-field> <y-name-field> <y-data-field>
```

Required arguments

<x-field>

**Syntax:** `<field>`

**Description:** Field to be used as the x-axis.

<y-name-field>

**Syntax:** `<field>`

**Description:** Field that contains the values to be used as labels for the data series.

<y-data-field>

**Syntax:** `<field>`
Description: Field that contains the data to be charted.

Usage

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

Basic example

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range All time when you run the search.

To show how to use the `untable` command, we need results that appear in a table format. Run this search.

`sourcetype=access_* status=200 action=purchase | top categoryId`

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>categoryId</th>
<th>count</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRATEGY</td>
<td>806</td>
<td>30.495649</td>
</tr>
<tr>
<td>ARCADE</td>
<td>493</td>
<td>18.653046</td>
</tr>
<tr>
<td>TEE</td>
<td>367</td>
<td>13.885736</td>
</tr>
<tr>
<td>ACCESSORIES</td>
<td>348</td>
<td>13.166856</td>
</tr>
<tr>
<td>SIMULATION</td>
<td>246</td>
<td>9.307605</td>
</tr>
<tr>
<td>SHOOTER</td>
<td>245</td>
<td>9.269769</td>
</tr>
<tr>
<td>SPORTS</td>
<td>138</td>
<td>5.221339</td>
</tr>
</tbody>
</table>

The `top` command automatically adds the count and percent fields to the results.

For each categoryId, there are two values, the count and the percent. When you untable these results, there will be three columns in the output:

- The first column lists the category IDs
- The second column lists the type of calculation: count or percent
- The third column lists the values for each calculation

When you use the `untable` command to convert the tabular results, you must specify the categoryId field first. You can use any field name you want for the
type of calculation and the values. For example:

```
sourcetype=access_* status=200 action=purchase | top categoryId |
untabtable categoryId calculation value
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>categoryId</th>
<th>calculation</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRATEGY</td>
<td>count</td>
<td>806</td>
</tr>
<tr>
<td>STRATEGY</td>
<td>percent</td>
<td>30.495649</td>
</tr>
<tr>
<td>ARCADE</td>
<td>count</td>
<td>493</td>
</tr>
<tr>
<td>ARCADE</td>
<td>percent</td>
<td>18.653046</td>
</tr>
<tr>
<td>TEE</td>
<td>count</td>
<td>367</td>
</tr>
<tr>
<td>TEE</td>
<td>percent</td>
<td>13.885736</td>
</tr>
<tr>
<td>ACCESSORIES</td>
<td>count</td>
<td>348</td>
</tr>
<tr>
<td>ACCESSORIES</td>
<td>percent</td>
<td>13.166856</td>
</tr>
<tr>
<td>SIMULATION</td>
<td>count</td>
<td>246</td>
</tr>
<tr>
<td>SIMULATION</td>
<td>percent</td>
<td>9.307605</td>
</tr>
</tbody>
</table>

### Extended example

The `untabtable` command does exactly what the name says, it converts tabular information into individual rows of results. Suppose you have this search:

```
...
| table _time EventCode Message
```

The search produces these results:

<table>
<thead>
<tr>
<th>_time</th>
<th>EventCode</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>date-time1</td>
<td>4136</td>
<td>Too late now</td>
</tr>
<tr>
<td>date_time2</td>
<td>1234</td>
<td>I don't know</td>
</tr>
<tr>
<td>date_time3</td>
<td>3456</td>
<td>Too busy, ask again later</td>
</tr>
<tr>
<td>date_time4</td>
<td>1256</td>
<td>Everything is happening at once</td>
</tr>
<tr>
<td>date_time4</td>
<td>1257</td>
<td>And right now, as well</td>
</tr>
</tbody>
</table>

Notice that this set of events has duplicate values in the `_time` field for `date_time4`. We will come back to that in a moment.

Use the `untabtable` command to remove the tabular format.
Here are the results from the `untable` command:

```
<table>
<thead>
<tr>
<th>_time</th>
<th>FieldName</th>
<th>FieldValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>date-time1</td>
<td>EventCode</td>
<td>4136</td>
</tr>
<tr>
<td>date-time1</td>
<td>Message</td>
<td>Too late now</td>
</tr>
<tr>
<td>date_time2</td>
<td>EventCode</td>
<td>1234</td>
</tr>
<tr>
<td>date-time2</td>
<td>Message</td>
<td>I dont know</td>
</tr>
<tr>
<td>date_time3</td>
<td>EventCode</td>
<td>3456</td>
</tr>
<tr>
<td>date-time3</td>
<td>Message</td>
<td>Too busy, ask again later</td>
</tr>
<tr>
<td>date_time4</td>
<td>EventCode</td>
<td>1256</td>
</tr>
<tr>
<td>date-time4</td>
<td>Message</td>
<td>Everything is happening at once</td>
</tr>
<tr>
<td>date_time4</td>
<td>EventCode</td>
<td>1257</td>
</tr>
<tr>
<td>date-time4</td>
<td>Message</td>
<td>And right now, as well</td>
</tr>
</tbody>
</table>
```

Remember that the original set of events had duplicates for `date_time4`. If you want to process the events in some way and then put the events back together, you can avoid mixing the duplicate events by using the `streamstats` command.

Use the `streamstats` command to give each event a unique record number and use that unique number as the key field for the `untable` and `xyseries` commands.

For example, you can add the `streamstats` command to your original search.

```
... | table _time EventCode Message | streamstats count as recno
```

The search produces these results:

```
<table>
<thead>
<tr>
<th>recno</th>
<th>_time</th>
<th>EventCode</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>date-time1</td>
<td>4136</td>
<td>Too late now</td>
</tr>
<tr>
<td>2</td>
<td>date_time2</td>
<td>1234</td>
<td>I dont know</td>
</tr>
<tr>
<td>3</td>
<td>date_time3</td>
<td>3456</td>
<td>Too busy, ask again later</td>
</tr>
<tr>
<td>4</td>
<td>date_time4</td>
<td>1256</td>
<td>Everything is happening at once</td>
</tr>
<tr>
<td>5</td>
<td>date_time4</td>
<td>1257</td>
<td>And right now, as well</td>
</tr>
</tbody>
</table>
```

You can then add the `untable` command to your search.

```
... | table _time EventCode Message | streamstats count as recno | untable _time FieldName FieldValue
```

The search produces these results:
These events can be put back together by using the `xyseries` command. For example:

```
...| xyseries _time FieldName FieldValue
```

(Thanks to Splunk user DalJeanis for this example.)

**See also**

`xyseries`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the `untable` command.

**where**

**Description**

The `where` command uses eval-expressions to filter search results. These eval-expressions must be Boolean expressions, where the expression returns either true or false. The `where` command returns only the results for which the eval expression returns true.
Syntax

where <eval-expression>

**Required arguments**

eval-expression

**Syntax:**  
<eval-mathematical-expression> | <eval-concatenate-expression> | <eval-comparison-expression> | <eval-boolean-expression> | <eval-function-call>

**Description:** A combination of values, variables, operators, and functions that represent the value of your destination field. See Usage.

The syntax of the eval expression is checked before running the search, and an exception is thrown for an invalid expression.

The following table describes characteristics of eval expressions that require special handling:

<table>
<thead>
<tr>
<th>Expression characteristics</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field names starting with numeric characters</td>
<td>If the expression references a field name that starts with a numeric character, the field name must be surrounded by single quotation marks.</td>
<td>'5minutes'=&quot;late&quot;</td>
</tr>
<tr>
<td>This expression is a field name equal to a string value. Because the field starts with a numeric it must be enclosed in single quotations. Because the value is a string, it must be enclosed in double quotations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field names with non-alphanumeric characters</td>
<td>If the expression references a field name that contains non-alphanumeric characters, the field name must be surrounded by single quotation marks.</td>
<td>new=count+'server-1'</td>
</tr>
<tr>
<td>This expression could be interpreted as a mathematical equation, where the dash is interpreted as a minus sign. To avoid this, you must enclose the</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Expression characteristics | Description | Example
--- | --- | ---
Literal strings | If the expression references a literal string, the literal string must be surrounded by double quotation marks. | new="server-"+count

Usage

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

The `where` command uses the same expression syntax as the `eval` command. Also, both commands interpret quoted strings as literals. If the string is not quoted, it is treated as a field name. Because of this, you can use the `where` command to compare two different fields, which you cannot use the `search` command to do.

<table>
<thead>
<tr>
<th>Command</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where</td>
<td>`...</td>
<td>where foo=bar`</td>
</tr>
<tr>
<td>Search</td>
<td>`...</td>
<td>search foo=bar`</td>
</tr>
<tr>
<td>Where</td>
<td>`...</td>
<td>where foo=&quot;bar&quot;`</td>
</tr>
</tbody>
</table>
**Boolean expressions**

The order in which Boolean expressions are evaluated with the `where` command is:

1. Expressions within parentheses
2. NOT clauses
3. AND clauses
4. OR clauses

This evaluation order is different than the order used with the `search` command. The `search` command evaluates OR clauses before AND clauses.

**Functions**

You can use a wide range of functions with the `where` command. For general information about using functions, see Evaluation functions.

The following table lists the supported functions by type of function. Use the links in the table to learn more about each function, and to see examples.

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Supported functions and syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Comparison and Conditional functions</strong></td>
<td>case(X,&quot;Y&quot;,...)</td>
</tr>
<tr>
<td></td>
<td>cidrmatch(&quot;X&quot;,Y)</td>
</tr>
<tr>
<td></td>
<td>coalesce(X,...)</td>
</tr>
<tr>
<td></td>
<td>false()</td>
</tr>
<tr>
<td></td>
<td>if(X,Y,Z)</td>
</tr>
<tr>
<td></td>
<td>in(VALUE-LIST)</td>
</tr>
<tr>
<td></td>
<td>like(TEXT, PATTERN)</td>
</tr>
<tr>
<td></td>
<td>match(SUBJECT, &quot;REGEX&quot;)</td>
</tr>
<tr>
<td></td>
<td>null()</td>
</tr>
<tr>
<td></td>
<td>nullif(X,Y)</td>
</tr>
<tr>
<td></td>
<td>searchmatch(X)</td>
</tr>
<tr>
<td></td>
<td>true()</td>
</tr>
<tr>
<td></td>
<td>validate()</td>
</tr>
<tr>
<td><strong>Conversion functions</strong></td>
<td>printf(&quot;format&quot;,arguments)</td>
</tr>
<tr>
<td></td>
<td>tonumber(NUMSTR,BASE)</td>
</tr>
<tr>
<td></td>
<td>tostring(X,Y)</td>
</tr>
<tr>
<td><strong>Cryptographic functions</strong></td>
<td>md5(X)</td>
</tr>
<tr>
<td></td>
<td>sha1(X)</td>
</tr>
<tr>
<td></td>
<td>sha256(X)</td>
</tr>
<tr>
<td></td>
<td>sha512(X)</td>
</tr>
<tr>
<td><strong>Date and Time functions</strong></td>
<td>now()</td>
</tr>
<tr>
<td></td>
<td>relative_time(X,Y)</td>
</tr>
<tr>
<td></td>
<td>strftime(X,Y)</td>
</tr>
<tr>
<td></td>
<td>strptime(X,Y)</td>
</tr>
<tr>
<td></td>
<td>time()</td>
</tr>
<tr>
<td><strong>Informational functions</strong></td>
<td>isbool(X)</td>
</tr>
<tr>
<td></td>
<td>isint(X)</td>
</tr>
<tr>
<td></td>
<td>isnonnull(X)</td>
</tr>
<tr>
<td></td>
<td>isnull(X)</td>
</tr>
<tr>
<td></td>
<td>isnum(X)</td>
</tr>
<tr>
<td></td>
<td>isstr(X)</td>
</tr>
<tr>
<td></td>
<td>typeof(X)</td>
</tr>
<tr>
<td>Type of function</td>
<td>Supported functions and syntax</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Mathematical functions</td>
<td></td>
</tr>
<tr>
<td>abs(X)</td>
<td>floor(X)</td>
</tr>
<tr>
<td>ceiling(X)</td>
<td>ln(X)</td>
</tr>
<tr>
<td>exact(X)</td>
<td>log(X,Y)</td>
</tr>
<tr>
<td>exp(X)</td>
<td>pi()</td>
</tr>
<tr>
<td></td>
<td>pow(X,Y)</td>
</tr>
<tr>
<td></td>
<td>round(X,Y)</td>
</tr>
<tr>
<td></td>
<td>sigfig(X)</td>
</tr>
<tr>
<td></td>
<td>sqrt(X)</td>
</tr>
<tr>
<td>Multivalue eval functions</td>
<td></td>
</tr>
<tr>
<td>commands(X)</td>
<td>mvfilter(X)</td>
</tr>
<tr>
<td>mvappend(X,...)</td>
<td>mvfind(MVFIELD,&quot;REGEX&quot;)</td>
</tr>
<tr>
<td>mvcount(MVFIELD)</td>
<td>mvindex(MVFIELD,STARTINDEX,ENDINDEX)</td>
</tr>
<tr>
<td>mvdedup(X)</td>
<td>mvjoin(MVFIELD,STR)</td>
</tr>
<tr>
<td>mvrange(X,Y,Z)</td>
<td>mvsort(X,Y,Z)</td>
</tr>
<tr>
<td>mvzip(X,Y)</td>
<td>split(X,Y)</td>
</tr>
<tr>
<td>Statistical eval functions</td>
<td></td>
</tr>
<tr>
<td>max(X,...)</td>
<td>min(X,...)</td>
</tr>
<tr>
<td></td>
<td>random()</td>
</tr>
<tr>
<td>Text functions</td>
<td></td>
</tr>
<tr>
<td>len(X)</td>
<td>rtrim(X,Y)</td>
</tr>
<tr>
<td>lower(X)</td>
<td>spath(X,Y)</td>
</tr>
<tr>
<td>ltrim(X,Z)</td>
<td>substr(X,Y,Z)</td>
</tr>
<tr>
<td>replace(X,Y,Z)</td>
<td>trim(X,Y,Z)</td>
</tr>
<tr>
<td></td>
<td>upper(X)</td>
</tr>
<tr>
<td></td>
<td>urlencode(X)</td>
</tr>
<tr>
<td>Trigonometry and Hyperbolic functions</td>
<td></td>
</tr>
<tr>
<td>acos(X)</td>
<td>atan2(X,Y)</td>
</tr>
<tr>
<td>acosh(X)</td>
<td>atanh(X)</td>
</tr>
<tr>
<td>asin(X)</td>
<td>cosh(X)</td>
</tr>
<tr>
<td>asinh(X)</td>
<td>cos(X)</td>
</tr>
<tr>
<td>atan(X)</td>
<td>sinh(X)</td>
</tr>
<tr>
<td></td>
<td>tan(X)</td>
</tr>
<tr>
<td></td>
<td>tanh(X)</td>
</tr>
</tbody>
</table>

**Examples**

1. **Use the where command to match IP addresses or a subnet**

Return "CheckPoint" events that match the IP or is in the specified subnet.

    host="CheckPoint" | where like(src, "10.9.165.%") OR cidrmatch("10.9.165.0/25", dst)

2. **Use the where command to specify a calculation**

Return "physicsjobs" events with a speed is greater than 100.

    sourcetype=physicsjobs | where distance/time > 100
See also
eval, search, regex

Answers
Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the where command.

x11

Description
The x11 command removes the seasonal pattern in your time-based data series so that you can see the real trend in your data. This command has a similar purpose to the trendline command, but it uses the more sophisticated and industry popular X11 method.

The seasonal component of your time series data can be either additive or multiplicative, defined as the two types of seasonality that you can calculate with x11: add() for additive and mult() for multiplicative. See About time-series forecasting in the Search Manual.

Syntax
x11 [<type>] [<period>] (<fieldname>) [AS <newfield>]

Required arguments
<fieldname>
    Syntax: <field>
    Description: The name of the field to calculate the seasonal trend.

Optional arguments
<type>
    Syntax: add() | mult()
    Description: Specify the type of x11 to compute, additive or multiplicative.
    Default: mult()

<period>
**Syntax:** `<int>`  
**Description:** The period of the data relative to the number of data points, expressed as an integer between 5 and 1000. If the period is 7, the command expects the data to be periodic every 7 data points. If you omit this parameter, Splunk software calculates the period automatically. The algorithm does not work if the period is less than 5 and will be too slow if the period is greater than 1000.

**<newfield>**  
**Syntax:** `<string>`  
**Description:** Specify a field name for the output of the `x11` command.  
**Default:** None

**Examples**

**Example 1:** In this example, the type is the default `mult` and the period is 15. The field name specified is `count`.

```
index=download | timechart span=1d count(file) as count | x11 mult15(count)
```

Because span=1d, every data point accounts for 1 day. As a result, the period in this example is 15 days. You can change the syntax in this example to ... | x11 15(count) because the `mult` type is the default type.

**Example 2:** In this example, the type is `add` and the period is 20. The field name specified is `count`.

```
index=download | timechart span=1d count(file) as count | x11 add20(count)
```
See also

predict, trendline

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the x11 command.

xmlkv

Description

The xmlkv command automatically extracts fields from XML-formatted data. For example, if the XML contains the following in its _raw data: `<foo>bar</foo>`, foo is the key and bar is the value.

For JSON-formatted data, use the spath command.

Syntax

xmlkv maxinputs=<int>

Required arguments

maxinputs

Syntax: maxinputs=<int>
Description: The maximum number of inputs.

Usage

The xmlkv command is a distributable streaming command. See Command types.

Examples

Example 1: Automatically extract fields from XML tags.
**Example 2:** Extract the key-value pairs from the first ten thousand events.

```
... | xmlkv maxinputs=10000
```

**See also**

`extract`, `kvform`, `multikv`, `rex`, `spath`, `xpath`

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `xmlkv` command.

---

**xmlunescape**

**Description**

Un-escapes XML characters, including entity references such as `&`, `<`, and `>`, so that they return to their corresponding characters. For example, `&amp;` becomes `&`.

**Syntax**

```
xmlunescape maxinputs=<int>
```

**Required arguments**

- `maxinputs`  
  
  **Syntax:** `maxinputs=<int>`  
  **Description:** The maximum number of inputs.

**Examples**

**Example 1:** Un-escape all XML characters.

```
... | xmlunescape
```
**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the xmlunescape command.

**xpath**

**Description**

Extracts the xpath value from field and sets the outfile attribute.

**Syntax**

\[xpath [outfile=<field>] <xpath-string> [field=<field>] [default=<string>]\]

**Required arguments**

xpath-string

**Syntax:** <string>

**Description:** Specifies the XPath reference.

**Optional arguments**

field

**Syntax:** field=<field>

**Description:** The field to find and extract the referenced xpath value from.

**Default:** _raw

outfield

**Syntax:** outfield=<field>

**Description:** The field to write, or output, the xpath value to.

**Default:** xpath

default

**Syntax:** default=<string>

**Description:** If the attribute referenced in xpath doesn't exist, this specifies what to write to the outfile. If this isn't defined, there is no default value.
Usage

The xpath command is a distributable streaming command. See Command types.

The xpath command supports the syntax described in the Python Standard Library 19.7.2.2. Supported XPath syntax.

Examples

1. Extract values from a single element in _raw XML events

You want to extract values from a single element in _raw XML events and write those values to a specific field.

The _raw XML events look like this:

```xml
<foo>
  <bar nickname="spock">
  </bar>
</foo>
<foo>
  <bar nickname="scotty">
  </bar>
</foo>
<foo>
  <bar nickname="bones">
  </bar>
</foo>
```

Extract the nickname values from _raw XML events. Output those values to the name field.

```
sourcetype="xml" | xpath outfield=name "//bar/@name"
```

2. Extract multiple values from _raw XML events

Extract multiple values from _raw XML events

The _raw XML events look like this:

```xml
<DataSet xmlns=""
  <identity_id>3017669</identity_id>
  <instrument_id>912383KM1</instrument_id>
```
Extract the values from the identity_id element from the _raw XML events:

... | xpath outfield=identity_id "//DataSet/identity_id"

This search returns two results: identity_id=3017669 and identity_id=1037669.

To extract a combination of two elements, sname with a specific value and instrument_id, use this search:

... | xpath outfield=instrument_id "//DataSet[sname='BARC']/instrument_id"

Because you specify sname='BARC', this search returns one result: instrument_id=912383KM1.

3. Testing extractions from XML events

You can use the makeresults command to test xpath extractions.

You must add field=xml to the end of your search. For example:

| makeresults
| eval xml="<DataSet xmlns="">"
  <identity_id>1037669</identity_id>
  <instrument_id>219383KM1</instrument_id>
  <transaction_code>SEL</transaction_code>
  <sname>TARC</sname>
  <currency_code>USA</currency_code>
</DataSet>" | xpath outfield=identity_id "//DataSet/identity_id" field=xml
See also

extract, kvform, multikv, rex, spath, xmlkv

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the xpath command.

xsDisplayConcept

The `xsDisplayConcept` command is an extreme search command used with Splunk Enterprise Security.

For information about this command, see Extreme search commands in Administer Splunk Enterprise Security.

xsDisplayContext

The `xsDisplayContext` command is an extreme search command used with Splunk Enterprise Security.

For information about this command, see Extreme search commands in Administer Splunk Enterprise Security.

xsFindBestConcept

The `xsFindBestConcept` command is an extreme search command used with Splunk Enterprise Security.

For information about this command, see Extreme search commands in Administer Splunk Enterprise Security.

xsListConcepts

The `xsListConcepts` command is an extreme search command used with Splunk Enterprise Security.
For information about this command, see Extreme search commands in Administer Splunk Enterprise Security.

**xsListContexts**

The `xsListContexts` command is an extreme search command used with Splunk Enterprise Security.

For information about this command, see Extreme search commands in Administer Splunk Enterprise Security.

**xsUpdateDDContext**

The `xsUpdateDDContext` command is an extreme search command used with Splunk Enterprise Security.

For information about this command, see Extreme search commands in Administer Splunk Enterprise Security.

**xsWhere**

The `xsWhere` command is an extreme search command used with Splunk Enterprise Security.

For information about this command, see Extreme search commands in Administer Splunk Enterprise Security.

**xyseries**

**Description**

Converts results into a tabular format that is suitable for graphing. This command is the inverse of the `untable` command.
Syntax

xyseries [grouped=<bool>] <x-field> <y-name-field> <y-data-field>...
[sep=<string>] [format=<string>]

Required arguments

<x-field>
   Syntax: <field>
   Description: Field to be used as the x-axis.

<y-name-field>
   Syntax: <field>
   Description: Field that contains the values to be used as labels for the data series.

<y-data-field>
   Syntax: <field> | <field>, <field>, ...
   Description: Field(s) that contains the data to be charted.

Optional arguments

format
   Syntax: format=<string>
   Description: Used to construct output field names when multiple data series are used in conjunction with a split-by-field. format takes precedence over sep and lets you specify a parameterized expression with the stats aggregator and function ($AGG$) and the value of the split-by-field ($VALUE$).

grouped
   Syntax: grouped= true | false
   Description: If true, indicates that the input is sorted by the value of the <x-field> and multifile input is allowed.
   Default: false

sep
   Syntax: sep=<string>
   Description: Used to construct output field names when multiple data series are used in conjunctions with a split-by field. This is equivalent to setting format to $AGG$<sep>$VALUE$.
Usage

The `xyseries` command is a **distributable streaming command**, unless `grouped=true` is specified and then the `xyseries` command is a **transforming command**. See [Command types](#).

Alias

The alias for the `xyseries` command is `maketable`.

Examples

**Example 1**: Reformat the search results.

```
... | xyseries delay host_type host
```

**Example 2**: Refer to this walkthrough to see how you can combine `stats` and `eval` with the `xyseries` command to create a report on multiple data series.

See also

`untable`

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `xyseries` command.

3rd party custom commands

Welcome to the Search Reference. See the left navigation panel for links to the built-in search commands. If you don't find a command in the list, that command might be part of a third-party app or add-on. For information about commands contributed by apps and add-ons, see the documentation on Splunkbase.
Internal Commands

About internal commands

Internal search commands refer to a set of commands that are designed to be used in specific situations, typically at the direction and with guidance from Splunk Support. These commands might be removed, or updated and reimplemented differently, in future versions.

Consult your Splunk Administrator or Splunk Support before using any of these commands.

- collapse
- dump
- findkeywords
- mcatalog
- noop
- runshellscrip
- sendalert

collapse

The collapse command is an unsupported, experimental command.

Description

The collapse command condenses multifile results into as few files as the chunksize option allows. This command runs automatically when you use outputlookup and outputcsv commands.

Syntax

... | collapse [chunksize=<num>] [force=<bool>]

Optional arguments

chunksize

Syntax: chunksize=<num>
Description: Limits the number of resulting files.
**Default:** 50000

`force`

**Syntax:** `force=<bool>`

**Description:** If `force=true` and the results are entirely in memory, re-divide the results into appropriated chunked files.

**Default:** `false`

**Examples**

**Example 1:** Collapse results.

```
... | collapse
```

## dump

**Description**

For Splunk Enterprise deployments, export search results to a set of chunk files on local disk. For information about other export methods, see Export search results in the *Search Manual*.

**Syntax**

dump basefilename=<string> [rollsize=<number>] [compress=<number>] [format=<string>] [fields=<comma-delimited-string>]

**Required arguments**

`basefilename`

**Syntax:** `basefilename=<string>`

**Description:** The prefix of the export filename.

**Optional arguments**

`compress`

**Syntax:** `compress=<number>`

**Description:** The gzip compression level. Specify a number from 0 to 9, where 0 means no compression and a higher number means more compression and slower writing speed.

**Default:** `2`
fields

**Syntax:** fields=<comma-delimited-string>

**Description:** A list of the fields to be exported. The entire list must be enclosed in quotation marks. Invalid field names are ignored.

format

**Syntax:** format= raw | csv | tsv | json | xml

**Description:** The output data format.

**Default:** raw

rollsize

**Syntax:** rollsize=<number>

**Description:** The minimum file size, in MB, at which point no more events are written to the file and it becomes a candidate for HDFS transfer.

**Default:** 63 MB

Usage

This command runs a specified search query and oneshot export search result to local disk at "$SPLUNK_HOME/var/run/splunk/dispatch/<sid>/dump". It recognizes a special field in the input events, _dstpath, which if set will be used as a path to be appended to dst to compute final destination path.

The `dump` command preserves the order of events as the events are received by the command.

Examples

**Example 1:** Export all events from index "bigdata" to the location "YYYYmmdd/HH/host" at "$SPLUNK_HOME/var/run/splunk/dispatch/<sid>/dump/" directory on local disk with "MyExport" as the prefix of export filenames. Partitioning of the export data is achieved by eval preceeding the dump command.

```
index=bigdata | eval _dstpath=strftime(_time, "%Y%m%d/%H") + "/" + host 
| dump basefilename=MyExport
```

**Example 2:** Export all events from index "bigdata" to the local disk with "MyExport" as the prefix of export filenames.

```
index=bigdata | dump basefilename=MyExport
```
Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the dump command.

findkeywords

Description

Given some integer labeling of events into groups, finds searches to generate these groups.

Syntax

findkeywords labelfield=<field>

Required arguments

labelfield

Syntax: labelfield=<field>

Description: A field name.

Usage

Use the findkeywords command after the cluster command, or a similar command that groups events. The findkeyword command takes a set of results with a field (labelfield) that supplies a partition of the results into a set of groups. The command derives a search to generate each of these groups. This search can be saved as an event type.

Examples

Return logs for specific log_level values and group the results

Return all logs where the log_level is DEBUG, WARN, ERROR, FATAL and group the results by cluster count.

```
index=_internal source=*splunkd.log* log_level!=info | cluster showcount=t | findkeywords labelfield=cluster_count
```
The result is a statistics table:

<table>
<thead>
<tr>
<th>field</th>
<th>value1</th>
<th>value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupID</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>cluster_count</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The values of `groupID` are the values of `cluster_count` returned from the `cluster` command.

See also

`cluster`, `findtypes`

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has about using the `findkeywords` command.

mcatalog

Description

The `mcatalog` command performs aggregations on the values in the `metric_name` and `dimension` fields in the metric indexes.

Syntax

```
| mcatalog [prestats=<bool>] [append=<bool>] ( <values"("<field> ")"> [AS <field>] ) [WHERE <logical-expression>] [ (BY|GROUPBY) <field-list> ]
```
Required arguments

values (<field>)

**Syntax:** values(<field>) [AS <field>]

**Description:** Returns the list of all distinct values of the specified field as a multivalue entry. The order of the values is lexicographical. See Usage.

Optional arguments

append

**Syntax:** append=<bool>

**Description:** Valid only when prestats=true. This argument runs the mcatalog command and adds the results to an existing set of results instead of generating new results.

**Default:** false

<field-list>

**Syntax:** <field>, ...

**Description:** Specify one or more fields to group results.

<logical-expression>

**Syntax:** <time-opts>|<search-modifier>|(NOT)?<logical-expression>|<index-expression>|<comparison-expression>|<(NOT)?<logical-expression>(OR)?<logical-expression>

**Description:** Includes time and search modifiers, comparison, and index expressions. Does not support CASE or TERM directives. You also cannot use the WHERE clause to search for terms or phrases.

prestats

**Syntax:** prestats=true | false

**Description:** Specifies whether to use the prestats format. The prestats format is a Splunk internal format that is designed to be consumed by commands that generate aggregate calculations. When using the prestats format you can pipe the data into the chart, stats, or timechart commands, which are designed to accept the prestats format. When prestats=true, AS instructions are not relevant. The field names for the aggregates are determined by the command that consumes the prestats format and produces the aggregate output.

**Default:** false
**Logical expression options**

<comparison-expression>
   **Syntax:** <field><comparison-operator><value> | <field> IN (<value-list>)
   **Description:** Compare a field to a literal value or provide a list of values that can appear in the field.

<index-expression>
   **Syntax:** "<string>" | <term> | <search-modifier>
   **Description:** Describe the events you want to retrieve from the index using literal strings and search modifiers.

<time-opts>
   **Syntax:** [<timeformat>] (<time-modifier>)*
   **Description:** Describe the format of the starttime and endtime terms of the search.

**Comparison expression options**

<comparison-operator>
   **Syntax:** = | != | < | <= | > | >=
   **Description:** You can use comparison operators when searching field/value pairs. Comparison expressions with the equal ( = ) or not equal ( != ) operator compare string values. For example, "1" does not match "1.0". Comparison expressions with greater than or less than operators < > <= >= numerically compare two numbers and lexicographically compare other values. See Usage.

<field>
   **Syntax:** <string>
   **Description:** The name of a field.

<value>
   **Syntax:** <literal-value>
   **Description:** In comparison-expressions, the literal number or string value of a field.

<value-list>
   **Syntax:** (<literal-value>, <literal-value>, ...)
   **Description:** Used with the IN operator to specify two or more values. For example use error IN (400, 402, 404, 406) instead of error=400 OR error=402 OR error=404 OR error=406
**Index expression options**

<string>
   **Syntax:** "<string>"
   **Description:** Specify keywords or quoted phrases to match. When searching for strings and quoted strings (anything that's not a search modifier), Splunk software searches the _raw field for the matching events or results.

<search-modifier>
   **Syntax:** <sourcetype-specifier> | <host-specifier> | <source-specifier> | <splunk_server-specifier>
   **Description:** Search for events from specified fields. For example, search for one or a combination of hosts, sources, and source types. See searching with default fields in the *Knowledge Manager manual*.

   <sourcetype-specifier>
      **Syntax:** sourcetype=<string>
      **Description:** Search for events from the specified sourcetype field.

   <host-specifier>
      **Syntax:** host=<string>
      **Description:** Search for events from the specified host field.

   <source-specifier>
      **Syntax:** source=<string>
      **Description:** Search for events from the specified source field.

   <splunk_server-specifier>
      **Syntax:** splunk_server=<string>
      **Description:** Search for events from a specific server. Use "local" to refer to the search head.

**Time options**

For a list of time modifiers, see *Time modifiers for search*.

<timeformat>
   **Syntax:** timeformat=<string>
   **Description:** Set the time format for starttime and endtime terms.
   **Default:** timeformat=%m/%d/%Y:%H:%M:%S.

<time-modifier>
Syntax: starttime=<string> | endtime=<string> | earliest=<time_modifier> | latest=<time_modifier>

Description: Specify start and end times using relative or absolute time.

Note: You can also use the earliest and latest attributes to specify absolute and relative time ranges for your search. For more about this time modifier syntax, see About search time ranges in the Search Manual.

starttime

Syntax: starttime=<string>

Description: Events must be later or equal to this time. Must match timeformat.

endtime

Syntax: endtime=<string>

Description: All events must be earlier or equal to this time.

Usage

You use the mcatalog command to search metrics data. The metrics data uses a specific format for the metrics fields. See Metrics data format in Metrics. The _values field is not allowed with this command.

The mcatalog command is a generating command for reports. Generating commands use a leading pipe character. The mcatalog command must be the first command in a search pipeline, except when append=true.

If your role does not have the list_metrics_catalog capability, you cannot use mcatalog.

See About defining roles with capabilities in the Securing Splunk Enterprise manual.

WHERE

Use the WHERE clause to filter by supported dimensions.

If you do not specify an index name in the WHERE clause, the mcatalog command returns results from the default metrics indexes associated with your role. If you do not specify an index name and you have no default metrics indexes associated with your role, mcatalog returns no results. To search against all metrics indexes use WHERE index=*. 723
For more information about defining default metrics indexes for a role, see Add and edit roles with Splunk Web in *Securing Splunk Enterprise*.

**Group by**

You can group by `dimension` and `metric_name` fields.

The `mcatalog` command does not allow grouping by time ranges. The argument is not included in its syntax.

**Time dimensions**

The `mcatalog` command does not recognize the following time-related dimensions.

<table>
<thead>
<tr>
<th>Unsupported dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>date_hour</code></td>
</tr>
<tr>
<td><code>date_mday</code></td>
</tr>
<tr>
<td><code>date_minute</code></td>
</tr>
<tr>
<td><code>date_month</code></td>
</tr>
<tr>
<td><code>date_second</code></td>
</tr>
<tr>
<td><code>date_wday</code></td>
</tr>
<tr>
<td><code>date_year</code></td>
</tr>
<tr>
<td><code>date_zone</code></td>
</tr>
<tr>
<td><code>metric_timestamp</code></td>
</tr>
<tr>
<td><code>time</code></td>
</tr>
<tr>
<td><code>timeendpos</code></td>
</tr>
<tr>
<td><code>timestamp</code></td>
</tr>
<tr>
<td><code>timestartpos</code></td>
</tr>
</tbody>
</table>

**Lexicographical order**

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.

- Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically as 10, 100, 70, 9.
- Uppercase letters are sorted before lowercase letters.
- Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

You can specify a custom sort order that overrides the lexicographical order. See the blog *Order Up! Custom Sort Orders*.  

724
Examples

1. **Return all of the metric names in a specific metric index**

   Return all of the metric names in the new-metric-idx.

   ```
   | mcatalog values(metric_name) WHERE index=new-metric-idx
   ```

2. **Return all metric names in the default metric indexes associated with the role of the user**

   If the user role has no default metric indexes assigned to it, the search returns no events.

   ```
   | mcatalog values(metric_name)
   ```

3. **Return all IP addresses for a specific metric_name among all metric indexes**

   Return of the IP addresses for the login.failure metric name.

   ```
   | mcatalog values(ip) WHERE index=* AND metric_name=login.failure
   ```

4. **Returns a list of all available dimensions in the default metric indexes associated with the role of the user**

   ```
   | mcatalog values(_dims)
   ```

**noop**

**Description**

The noop command is an internal command that you can use to debug your search. It includes several arguments that you can use to troubleshoot search optimization issues.

You cannot use the noop command to add comments to a search. If you are looking for a way to add comments to your search, see Add comments to searches in the Search Manual.
Syntax

noop [newfield=<string>] [log_<level> = <channel-list>] [search_optimization = <boolean>] [search_optimization.<optimization_type> = <boolean>]...

Required arguments

None.

Optional arguments

newfield

Syntax: <string>
Description: The name of a new field where you want the results placed.

log_<level>

Syntax: log_<level> = <channel-list>
Description: Lets you set or change the log levels for log channels at search startup. Valid values for <level> are the Splunk platform internal logging levels: debug, info, warn, and error, and fatal. You can apply different log levels to different sets of channels.

<channel-list>

Syntax: <channel>, ...
Description: Specify one or more log channels to apply the log level to. Use wildcards to catch all channels with a matching string of characters in their name.

search_optimization

Syntax: search_optimization = <boolean>
Description: Enables or disables all optimizations for the search.
Default: true

optimization_type

Syntax: search_optimization.<optimization_type> = <boolean>
Description: Enables or disables a specific type of search optimization for the search. To disable multiple optimization types, create a comma-separated list of arguments.
Default: true

Here are the search_optimization.<optimization_type> arguments that you can use with noop.
<table>
<thead>
<tr>
<th>search_optimization argument</th>
<th>Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>search_optimization.eval_merge</td>
<td>Eval merge optimization</td>
</tr>
<tr>
<td>search_optimization.merge_union</td>
<td>Merge union optimization</td>
</tr>
<tr>
<td>search_optimization.predicate_merge</td>
<td>Predicate merge optimizations</td>
</tr>
<tr>
<td>search_optimization.predicate_push</td>
<td>Predicate pushdown optimizations</td>
</tr>
<tr>
<td>search_optimization.predicate_split</td>
<td>Predicate split optimizations</td>
</tr>
<tr>
<td>search_optimization.projection_elimination</td>
<td>Projection elimination optimizations</td>
</tr>
<tr>
<td>search_optimization.required_field_values</td>
<td>Required field value optimizations</td>
</tr>
<tr>
<td>search_optimization.search_sort_normalization</td>
<td>Predicate sort normalization</td>
</tr>
<tr>
<td>search_optimization.search_flip_normalization</td>
<td>Predicate flip normalization</td>
</tr>
<tr>
<td>search_optimization.replace_append_with_union</td>
<td>Replace append with union optimization</td>
</tr>
</tbody>
</table>

For more information about specific search optimization types, see Built-in optimizations.

**Usage**

You can use the `noop` command to enable or disable search optimizations when you run a search. Enabling or disabling search optimizations can help you troubleshoot certain kinds of search issues. For example, you might experiment with disabling and enabling search optimizations to determine whether they are causing a search to be slow to complete.

For information about managing search optimization through `limits.conf` for all of the users in your Splunk platform deployment, see Built-in optimization in the *Search Manual*. 
Managing all search optimizations with the noop command

The `noop` command can enable or disable all search optimizations for a single run of a search.

If all search optimizations are enabled for your Splunk deployment in `limits.conf`, you can add the following argument to the end of a search string to disable all optimizations when you run that search:

```plaintext
.... | noop search_optimization=false
```

If all search optimizations are disabled for your Splunk deployment in `limits.conf`, you can add the following argument to the end of a search string to enable all search optimizations when you run that search:

```plaintext
.... | noop search_optimization=true
```

Managing specific search optimizations with the noop command

You can use the `optimization_type` argument to selectively disable or enable specific types of search optimization.

Here is an example of a set of `noop` arguments that disable the predicate merge and predicate pushdown optimizations for a search.

```plaintext
.... | noop search_optimization.predicate_merge=false,
     search_optimization.predicate_push=false
```

This example works only if you have enabled all optimizations in `limits.conf`.

When you set `enabled=false` for the `[search_optimization]` stanza in `limits.conf` you disable all search optimizations for your Splunk platform deployment. With this `limits.conf` configuration, your searches can use `noop` to enable all optimizations and selectively disable specific optimization types.

For example, if you have the `[search_optimization]` stanza set to `enabled=false` in `limits.conf`, the following search enables all optimizations except projection elimination.

```plaintext
index=_internal | eval c = x * y / z | stats count BY a, b | noop
                  search_optimization=true,
                  search_optimization.projection_elimination=false
```
However, when you set `enabled=false` for the `[search_optimization]` stanza in `limits.conf`, your searches cannot enable specific optimization types unless specific conditions are met. See How `noop` interoperates with `limits.conf` search optimization settings.

**How the `noop` command interoperates with `limits.conf` search optimization settings**

Review how you have configured search optimization for your Splunk platform deployment in `limits.conf` before you use the `noop` command to enable or disable optimization types. The search processor respects `limits.conf` settings for optimization types only when `[search_optimization]` is enabled.

For example, if the `[search_optimization]` stanza is set to `enabled=true` in `limits.conf`, the search processor checks whether individual optimization types are enabled or disabled in `limits.conf`. On the other hand, if the `[search_optimization]` stanza is set to `enabled = false`, the search processor does not check the settings for other optimization types. It assumes all of the optimization types are set to `enabled=false`.

This search processor logic affects the way that the `noop` command works when you use it to enable or disable search optimization for an individual search.

For example, imagine that you have the following configuration in `limits.conf`:

```plaintext
[search_optimization]
enabled=false

[search_optimization::projection_elimination]
enabled=false
```

With this configuration, the search processor ignores the disabled projection elimination optimization. Because `[search_optimization]` is disabled, the search processor assumes all optimizations are disabled.

Say you have this configuration, and you run the following search, which uses the `noop` command to enable search optimization:

```plaintext
.... | noop search_optimization=true
```

When you do this, you enable search optimization, but the search processor sees that in `limits.conf`, the projection elimination optimization is disabled. It runs the search with all optimization types enabled except projection elimination.
Instead, use the `noop` command in a search to enable search optimization and selectively enable the projection elimination optimization:

```plaintext
.... | noop search_optimization=true
     search_optimization.projection_elimination=true
```

When this search runs, it overrides both `limits.conf` settings: the setting for `[search_optimization]` and the setting for `[search_optimization::projection_elimination]`. The search runs with all optimizations enabled.

**Use noop to set debugging channels for a search**

The `log_<level>` argument lets you set the debugging channel for a search at a specific log level, such as `debug` or `warn`. You might use this if you need to set the log level for a specific search but do not have CLI access to the search.

The Splunk platform changes the log level after it parses the `noop` command. This means that anything preceding the `noop` command in the search string uses the default log level. If you want the log level to be changed on the indexers, place the `noop` command in the streaming part of the search.

The `log<level>` argument supports wildcard matching. You can also set different log levels for different debugging channels in the same search.

```plaintext
.... | noop log_debug=Cache*
     log_info=SearchOperator:kv,SearchOperator:multikv
```

For more information about logs and setting log levels for debugging channels, see What Splunk logs about itself in the Troubleshooting Manual.

**runshellscript**

**Description**

For Splunk Enterprise deployments, executes scripted alerts. This command is not supported as a search command.

**Syntax**

```
runshellscript <script-filename> <result-count> <search-terms> <search-string>
<savedsearch-name> <description> <results-url> <deprecated-arg>
```
<results_file>

**Usage**

The script file needs to be located in either

```bash
$SPLUNK_HOME/etc/system/bin/scripts
```

OR

```bash
$SPLUNK_HOME/etc/apps/<app-name>/bin/scripts
```

The following table describes the arguments passed to the script. These arguments are not validated.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>The filename of the script.</td>
</tr>
<tr>
<td>$1</td>
<td>The result count, or number of events returned.</td>
</tr>
<tr>
<td>$2</td>
<td>The search terms.</td>
</tr>
<tr>
<td>$3</td>
<td>The fully qualified query string.</td>
</tr>
<tr>
<td>$4</td>
<td>The name of the saved search in Splunk.</td>
</tr>
<tr>
<td>$5</td>
<td>The description or trigger reason. For example, &quot;The number of events was greater than 1.&quot;</td>
</tr>
<tr>
<td>$6</td>
<td>The link to saved search results.</td>
</tr>
<tr>
<td>$7</td>
<td>DEPRECATED - empty string argument.</td>
</tr>
<tr>
<td>$8</td>
<td>The path to the results file, <code>results.csv</code>. The results file contains raw results.</td>
</tr>
</tbody>
</table>

**See also**

*script*

**Answers**

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the `runshellscript` command.

**sendalert**

**Description**

Use the `sendalert` command to invoke a custom alert action. The command gathers the configuration for the alert action from the `alert_actions.conf` file and the saved search and custom parameters passed using the command.
arguments. Then the command performs token replacement. The command
determines the alert action script and arguments to run, creates the alert action
payload and executes the script, handing over the payload by using STDIN to the
script process.

When running the custom script, the `sendalert` command honors the `maxtime`
setting from the `alert_actions.conf` file and terminates the process if the
process runs longer than the configured threshold. By default the threshold is set
to 5 minutes.

See "Advanced options for working with custom alert actions" in the Developing
Views and Apps for Splunk Web manual.

**Syntax**

```
sendalert <alert_action_name> [results_link=<url>] [results_path=<path>]
[param.<name>="value"...]```

**Required arguments**

`alert_action_name`

*Syntax:* `<alert_action_name>`

*Description:* The name of the alert action configured in the
`alert_actions.conf` file

**Optional arguments**

`results_link`

*Syntax:* `results_link=<url>`

*Description:* Set the URL link to the search results.

`results_path`

*Syntax:* `results_path=<path>`

*Description:* Set the location to the file containing the search results.

`param.<name>`

*Syntax:* `param.<name>="value"`

*Description:* The parameter name and value. You can use this name and value pair
to specify a variety of things, such as a threshold value, a team name, or the text of a message.
Usage

When you use the sendalert command in an ad hoc search, the command might be called multiple times if there are a large number of search results. This occurs because previewing the search results on the Statistics tab is enabled by default. If you are using an ad hoc search to test the sendalert command, testing turn off preview to avoid the command being called multiple times.

When the sendalert command is included in a saved search, such as a scheduled report or a scheduled search, the command is called only one time.

Search results format

When the sendalert command is used in a search or in an alert action, the search results are stored in an archive file in the dispatch directory using the CSV format. The file name is results.csv.gz. The default format for the search results is SRS, a Splunk-specific binary format for the search results. The CSV format for the archive file is used so that scripts can process the results file. The default SRS format is not designed to be parsed by scripts.

The archived search results format is controlled through the forceCsvResults setting. This setting is in the [default] stanza in the alert_actions.conf file.

Examples

Example 1: Invoke an alert action without any arguments. The alert action script handles checking whether there are necessary parameters that are missing and report the error appropriately.

```bash
... | sendalert myaction
```

Example 2: Trigger the hipchat custom alert action and pass in room and message as custom parameters.

```bash
... | sendalert hipchat param.room="SecOps" param.message="There is a security problem!"
```

Example 3: Trigger the servicenow alert option.

```bash
... | sendalert servicenow param.severity="3" param.assigned_to="DevOps" param.short_description="Splunk Alert: this is a potential security issue"
```
Search in the CLI

About searches in the CLI

If you use Splunk Enterprise, you can issue search commands from the command line using the Splunk CLI. This topic discusses how to search from the CLI. If you're looking for how to access the CLI and find help for it, refer to "About the CLI" in the Admin manual.

CLI help for search

You can run historical searches using the search command, and real-time searches using the rtsearch command. The following is a table of useful search-related CLI help objects. To see the full help information for each object, type into the CLI:

./splunk help <object>

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtsearch</td>
<td>Returns the parameters and syntax for real-time searches.</td>
</tr>
<tr>
<td>search</td>
<td>Returns the parameters and syntax for historical searches.</td>
</tr>
<tr>
<td>search-commands</td>
<td>Returns a list of search commands that you can use from the CLI.</td>
</tr>
<tr>
<td>search-fields</td>
<td>Returns a list of default fields.</td>
</tr>
<tr>
<td>search-modifiers</td>
<td>Returns a list of search and time-based modifiers that you can use to narrow your search.</td>
</tr>
</tbody>
</table>

Search in the CLI

Historical and real-time searches in the CLI work the same way as searches in Splunk Web, except that there is no timeline rendered with the search results and there is no default time range. Instead, the results are displayed as a raw events list or a table, depending on the type of search.

- For more information, read "Type of searches" in the Search Overview chapter of the Search Manual.
The syntax for CLI searches is similar to the syntax for Splunk Web searches, except that you can pass parameters outside of the query to specify the time limit of the search, where to run the search, and how results are displayed.

- For more information about the CLI search options, see the next topic in this chapter, "CLI search syntax".
- For more information about how to search remote Splunk servers from your local server, see "Access and use the CLI on a remote server" in the Admin manual.

**Syntax for searches in the CLI**

This is a quick discussion of the syntax and options available for using the `search` and `rtsearch` commands in the CLI.

The syntax for CLI searches is similar to the syntax for searches you run from Splunk Web except that you can pass parameters outside of the search object to control the time limit of the search, specify the server where the search is to be run, and specify how results are displayed.

```
search | rtsearch [object][,-parameter <value>]
```

**Search defaults**

By default when you run a search from the CLI, the search is uses All Time as the time range. You can specify time ranges using one of the CLI search parameters, such as `earliest_time`, `index_earliest`, or `latest_time`.

The first 100 events are returned when you run a historical search using the CLI. Use the `maxout` search parameter to specify the number of events to return.

**Search objects**

Search objects are enclosed in single quotes (' ') and can be keywords, expressions, or a series of search commands. On Windows OS use double quotes (" ") to enclose your search object.

- For more information about searching, see the "Start searching" topic in the Search Tutorial.
- For a brief description of every search command, see the "Command quick reference" in the Search Reference.
- For a quick reference for Splunk concepts, features, search commands, and functions, see the "Quick Reference Guide" in the Search Reference.

Search objects can include not only keywords and search commands but also fields and modifiers to specify the events you want to retrieve and the results you want to generate.

- For more information about fields, see the "Use fields to search" topic in the Search Tutorial.
- For more information about default fields and how to use them, see the "Use default and internal fields" topic in the Knowledge Manager Manual.
- For more information about time modifiers, see the "Time modifiers for search" topic in the Search Reference Manual.

**Search parameters**

Search parameters are options that control the way the search is run or the way the search results are displayed. All of these parameters are optional. Parameters that take Boolean values support \{0, false, f, no\} as negatives and \{1, true, t, yes\} positives.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Defaults</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>(&lt;app&gt;)</td>
<td>(\text{search})</td>
<td>Specify the name of the app in which to run your search.</td>
</tr>
<tr>
<td>batch</td>
<td>(&lt;\text{bool}&gt;)</td>
<td>F</td>
<td>Indicates how to handle updates in preview mode.</td>
</tr>
<tr>
<td>detach</td>
<td>(&lt;\text{bool}&gt;)</td>
<td>F</td>
<td>Triggers an asynchronous search and displays the job ID and TTL for the search.</td>
</tr>
<tr>
<td>earliest_time</td>
<td>(&lt;\text{time-modifier}&gt;)</td>
<td>-</td>
<td>The relative time modifier for the start time of the search.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Values</td>
<td>Defaults</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>header</td>
<td>&lt;bool&gt;</td>
<td>T</td>
<td>Indicates whether to display a header in the table output mode.</td>
</tr>
<tr>
<td>index_earliest</td>
<td>&lt;time-modifer&gt;</td>
<td></td>
<td>The start time of the search. This can be expressed as an epoch or a relative time modifier and uses the same syntax as the &quot;earliest&quot; and &quot;latest&quot; time modifiers for the search language. This is optional for both search and rtsearch.</td>
</tr>
<tr>
<td>index_latest</td>
<td>&lt;time-modifer&gt;</td>
<td></td>
<td>The end time of the search. This can be expressed as an epoch or a relative time modifier and uses the same syntax as the &quot;earliest&quot; and &quot;latest&quot; time modifiers for the search language. This is optional for both search and rtsearch.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Values</td>
<td>Defaults</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>latest_time</td>
<td>&lt;time-modifer&gt;</td>
<td>--</td>
<td>The relative time modifier for the end time of the search. For search, if not specified, defaults to the end of the time (or the time of the last event in the data) so that any &quot;future&quot; events are also included. For rtsearch, this is a required parameter and the real-time search will not run if it's not specified.</td>
</tr>
<tr>
<td>max_time</td>
<td>&lt;number&gt;</td>
<td>0</td>
<td>The length of time in seconds that a search job runs before it is finalized. A value of 0 means that there is no time limit.</td>
</tr>
<tr>
<td>maxout</td>
<td>&lt;number&gt;</td>
<td>search, 100 rtsearch, 0</td>
<td>The maximum number of events to return or send to...</td>
</tr>
<tr>
<td>Parameter</td>
<td>Values</td>
<td>Defaults</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>output</td>
<td>rawdata, table, csv, auto</td>
<td></td>
<td>Use <code>rawdata</code> for non-transforming searches. Use <code>table</code> for transforming searches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Indicates how to display the job.</td>
</tr>
<tr>
<td>preview</td>
<td>&lt;bool&gt;</td>
<td>T</td>
<td>Indicates that reporting searches should be previewed (displayed as results are calculated).</td>
</tr>
<tr>
<td>timeout</td>
<td>&lt;number&gt;</td>
<td>0</td>
<td>The length of time in seconds that a search job is allowed to live after running. A value of 0 means that the job is canceled immediately after it is run.</td>
</tr>
<tr>
<td>uri</td>
<td>[http</td>
<td>https]://name_of_server:management_port</td>
<td></td>
</tr>
</tbody>
</table>

stdout when exporting events. A value of 0 means that it will output an unlimited number of events.
The default uri value is the mgmtHostPort value that you defined in the Splunk server's web.conf.

For more information, see Access and use the CLI on a remote Splunk Server in the Admin manual.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Defaults</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrap</td>
<td>&lt;bool&gt;</td>
<td>T</td>
<td>Indicates whether to wrap for individual lines that are longer than the terminal width.</td>
</tr>
</tbody>
</table>

**Examples**

You can see more examples in the CLI help information.

**Example 1:** Retrieve events from yesterday that match root sessions.

```
./splunk search "session root daysago=1"
```

**Example 2:** Retrieve events that match web access errors and detach the search.

```
```
Example 3: Run a windowed real-time search.

```
./splunk rtsearch 'index=_internal' -earliest_time 'rt-30s' -latest_time 'rt+30s'
```

See more examples of Real-time searches and reports in the CLI in the Admin manual.

Example 4: Return a list of unique hostnames.

There are two recommended ways that you can do this. This first is with the stats command:

```
./splunk search 'index=* | stats count by host | fields - count' -preview true
```

Alternatively, since you are only interested in the host field, you can use the metadata command:

```
./splunk search '| metadata type=hosts | fields host' -preview true
```

Here, the -preview flag is optional and used to view the results as it is returned. In contrast, the table command, unlike the fields command, generally requires all inputs before it can emit any non-preview output. In this case, you would need to use the preview flag to be able to view the results of the search.

Example 5: Return yesterday's internal events.

```
./splunk search 'index=_internal' -index_earliest -1d@d -index_latest @d
```