

Splunk 5.0.1

Search Reference

Generated: 1/18/2013 9:55 am

Table of Contents

Introduction.....	1
Welcome to the Search Reference Manual.....	1
How to use this manual.....	1
Search Reference Overview.....	3
Search Command Cheat Sheet and Search Language Quick Reference Card.....	3
Popular search commands.....	11
Splunk for SQL users.....	13
Search Commands and Functions.....	20
All search commands.....	20
Functions for eval and where.....	27
Functions for stats, chart, and timechart.....	37
Common date and time format variables.....	41
Time modifiers for search.....	43
List of data types.....	47
Search Command Reference.....	83
abstract.....	83
accum.....	84
addcoltotals.....	85
addinfo.....	86
addtotals.....	87
analyzefields.....	89
anomalies.....	90
anomalousvalue.....	94
append.....	97
appendcols.....	101
appendpipe.....	103
associate.....	103
audit.....	106
autoregress.....	107
bucket.....	109
bucketdir.....	112
chart.....	113
cluster.....	127
collect.....	131
concurrency.....	133

Table of Contents

Search Command Reference

contingency.....	138
convert.....	143
correlate.....	148
crawl.....	150
dbinspect.....	151
dedup.....	153
delete.....	156
delta.....	157
diff.....	162
erex.....	164
eval.....	165
eventcount.....	174
eventstats.....	175
extract (kv).....	177
fieldformat.....	179
fields.....	180
fieldsummary.....	182
filldown.....	183
fillnull.....	184
findtypes.....	185
folderize.....	187
format.....	188
gauge.....	189
gentimes.....	191
head.....	193
highlight.....	194
history.....	195
iconify.....	196
input.....	197
inputcsv.....	198
inputlookup.....	200
iplocation.....	202
join.....	203
kmeans.....	205
kvform.....	207
loadjob.....	208
localize.....	210
localop.....	212

Table of Contents

Search Command Reference

lookup.....	212
makecontinuous.....	215
makemv.....	217
map.....	219
metadata.....	221
metasearch.....	223
multikv.....	225
multisearch.....	227
mvcombine.....	228
mvexpand.....	230
nomv.....	232
outlier.....	233
outputcsv.....	235
outputlookup.....	236
outputtext.....	238
overlap.....	239
predict.....	240
rangemap.....	243
rare.....	245
regex.....	247
relevancy.....	249
retime.....	249
rename.....	250
replace.....	252
rest.....	253
return.....	255
reverse.....	256
rex.....	257
rtorder.....	259
run.....	260
savedsearch.....	260
script.....	262
scrub.....	263
search.....	264
searchtxn.....	270
selfjoin.....	271
set.....	273
setfields.....	274

Table of Contents

Search Command Reference

sendemail.....	275
sichart.....	278
sirare.....	279
sistats.....	280
sitimechart.....	281
sitop.....	282
sort.....	283
spath.....	286
stats.....	291
strcat.....	299
streamstats.....	300
table.....	303
tags.....	307
tail.....	309
timechart.....	310
top.....	323
transaction.....	326
transpose.....	335
trendline.....	336
typeahead.....	338
typelearner.....	339
typer.....	340
uniq.....	341
untable.....	342
where.....	343
x11.....	344
xmlkv.....	346
xmlunescape.....	347
xpath.....	348
xyseries.....	350

Internal Search Commands.....352

About internal commands.....	352
collapse.....	352
dispatch.....	353
runshellscript.....	353
tscollect.....	354
tstats.....	356

Table of Contents

Search in the CLI.....	360
About searches in the CLI.....	360
Syntax for searches in the CLI.....	361

Introduction

Welcome to the Search Reference Manual

In this manual, you'll find 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.

If you're looking for an introduction to searching in Splunk, read the Search Manual to get you started.

See the ["List of search commands"](#) in the Search Overview chapter for a catalog of the search commands, with a short description of what they do and related search commands. Each search command links you to its reference page in the Search Command chapter of this manual. If you want to just jump right in and start searching, the [Search command cheat sheet](#) is a quick reference complete with descriptions and examples.

Before you continue, read ["How to use this manual"](#) for the conventions and rules used in this manual.

Make a PDF

If you'd like a PDF version of this manual, click the red **Download the Search Reference as PDF** link below the table of contents on the left side of this page. A PDF version of the manual is generated on the fly for you, and you can save it or print it out to read later.

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.

Layout for each topic

Each search command topic contains the following headers: synopsis, description, examples, and see also.

Synopsis

The synopsis includes a short description of each search command, the complete syntax for each search command, and a description for each argument. If the arguments have another hierarchy of options, each of these sets of options follow the argument descriptions.

Required arguments

The list of required parameters and their syntax.

Optional arguments

The list of optional parameters and their syntax.

Description

The description includes details about how to use the search command.

Examples

This section lists examples of usage for the search command.

See also

This sections lists and links to all related or similar search commands.

Conventions used to describe syntax

The syntax for each search command is defined under the "Synopsis". The arguments are presented in the syntax in the order they are meant to be used.

Conventions used to describe arguments

Arguments are either Required or Optional and are listed alphabetically under their respective subheadings. For each argument, there is a **Syntax** and **Description** part. The description includes usage information and defaults.

Search Reference Overview

Search Command Cheat Sheet and Search Language Quick Reference Card

The Search Command Cheat Sheet is a quick command reference complete with descriptions and examples. The Search Command Cheat Sheet is also available for download as an eight-page PDF file.

The Search Language Quick Reference Card, available only as a PDF file, is a six-page reference card that provides fundamental search concepts, commands, functions, and examples.

Note: In the examples on this page, a leading ellipsis (...) indicates that there is a search before the pipe operator. A leading pipe | prevents the CLI or UI from prepending the "search" operator on your search.

Answers

Have questions about search commands? Check out Splunk Answers to see what questions and answers other Splunk users had about the search language. Now, on to the cheat sheet!

administrative

View information in the "audit" index.	<code>index=_audit audit</code>
Crawl root and home directories and add all possible inputs found (adds configuration information to "inputs.conf").	<code> crawl root="/;/Users/" input add</code>
Display a chart with the span size of 1 day.	<code> dbinspect index=_internal span=1d</code>
Return the values of "host" for events in the "_internal" index.	<code> metadata type=hosts index=_internal</code>
Return typeahead information for sources in the "_internal" index.	<code> typeahead prefix=source count=10 index=_internal</code>

alerting

Send search results to the specified email.	<code>... sendemail to="elvis@splunk.com"</code>
---	--

fields

add

Save the running total of "count" in a field called "total_count".	... accum count AS total_count
Add information about the search to each event.	... addinfo
Search for "404" events and append the fields in each event to the previous search results.	... appendcols [search 404]
For each event where 'count' exists, compute the difference between count and its previous value and store the result in 'countdiff'.	... delta count AS countdiff
Extracts out values like "7/01", putting them into the "monthday" attribute.	... erex monthday examples="7/01"
Set velocity to distance / time.	... eval velocity=distance/time
Extract field/value pairs and reload field extraction settings from disk.	... extract reload=true
Extract field/value pairs that are delimited by " ," and values of fields that are delimited by "=:".	... extract pairdelim=" ," kvdelim="=:", auto=f
Add location information (based on IP address).	... iplocation
Extract values from "eventtype.form" if the file exists.	... kvform field=eventtype
Extract the "COMMAND" field when it occurs in rows that contain "splunkd".	... multikv fields COMMAND filter splunkd
Set <code>range</code> to "green" if the <code>date_second</code> is between 1-30; "blue", if between 31-39; "red", if between 40-59; and "gray", if no range matches (for example, if <code>date_second=0</code>).	... rangemap field=date_second green=1-30 blue=31-39 red=40-59 default=gray
Calculate the relevancy of the search and sort the results in descending order.	disk error relevancy sort -relevancy
Extract "from" and "to" fields using regular expressions. If a raw event contains "From: Susan To: Bob", then from=Susan and to=Bob.	... rex field=_raw "From: (?<from>.*) To: (?<to>.*)"
Extract the "author" field from XML or JSON formatted data about books.	... spath output=author path=book{@author}
Add the field: "comboIP". Values of "comboIP" = "sourceIP" + "/" + "destIP".	... strcat sourceIP "/" destIP comboIP
Extract field/value pairs from XML formatted data. "xmlkv" automatically extracts values between XML tags.	... xmlkv

convert

Convert every field value to a number value except for values in the field "foo" (use the "none" argument to specify fields to ignore).	<code>... convert auto(*) none(foo)</code>
Change all memory values in the "virt" field to Kilobytes.	<code>... convert memk(virt)</code>
Change the sendmail syslog duration format (D+HH:MM:SS) to seconds. For example, if "delay="00:10:15"", the resulting value will be "delay="615"".	<code>... convert dur2sec(delay)</code>
Convert values of the "duration" field into number value by removing string values in the field value. For example, if "duration="212 sec"", the resulting value will be "duration="212"".	<code>... convert rmunit(duration)</code>
Separate the value of "foo" into multiple values.	<code>... makemv delim=":" allowempty=t foo</code>
For sendmail events, combine the values of the senders field into a single value; then, display the top 10 values.	<code>eventtype="sendmail" nomv senders top senders</code>

filter

Keep the "host" and "ip" fields, and display them in the order: "host", "ip".	<code>... fields + host, ip</code>
Remove the "host" and "ip" fields.	<code>... fields - host, ip</code>

modify

Build a time series chart of web events by host and fill all empty fields with NULL.	<code>sourcetype="web" timechart count by host fillnull value=NULL</code>
Rename the "_ip" field as "IPAddress".	<code>... rename _ip as IPAddress</code>
Change any host value that ends with "localhost" to "localhost".	<code>... replace *localhost with localhost in host</code>

read

There is a lookup table specified in a stanza name 'usertogroup' in transforms.conf. This lookup table contains (at least) two fields, 'user' and 'group'. For each event, we look up the value of the field 'local_user' in the table and for any entries that matches, the value of the 'group' field in the lookup table will be written to the field 'user_group' in the event.	<code>... lookup usertogroup user as local_user OUTPUT group as user_group</code>
---	---

formatting

Show a summary of up to 5 lines for each search result.	<code>... abstract maxlines=5</code>
Compare the "ip" values of the first and third search results.	<code>... diff pos1=1 pos2=3 attribute=ip</code>
Highlight the terms "login" and "logout".	<code>... highlight login,logout</code>
Displays an different icon for each eventtype.	<code>... iconify eventtype</code>
Output the "_raw" field of your current search into "_xml".	<code>... outputtext</code>
Anonymize the current search results.	<code>... scrub</code>
Un-escape all XML characters.	<code>... xmlunescape</code>

index

add

Add each source found by crawl in the default index with automatic source classification (sourcetying)	<code> crawl input add</code>
--	----------------------------------

delete

Delete events from the "imap" index that contain the word "invalid"	<code>index=imap invalid delete</code>
---	--

summary

Put "download" events into an index named "downloadcount".	<code>eventtypetag="download" collect index=downloadcount</code>
Find overlapping events in "summary".	<code>index=summary overlap</code>
Compute the necessary information to later do 'chart avg(foo) by bar' on summary indexed results.	<code>... sichart avg(foo) by bar</code>
Compute the necessary information to later do 'rare foo bar' on summary indexed results.	<code>... sirare foo bar</code>
Compute the necessary information to later do 'stats avg(foo) by bar' on summary indexed results	<code>... sistats avg(foo) by bar</code>
Compute the necessary information to later do 'timechart avg(foo) by bar' on summary indexed results.	<code>... sitimechart avg(foo) by bar</code>
Compute the necessary information to later do 'top foo bar' on summary indexed results.	<code>... sitop foo bar</code>

reporting

Calculate the sums of the numeric fields of each result, and put the sums in the field "sum".	<code>... addtotals fieldname=sum</code>
Analyze the numerical fields to predict the value of "is_activated".	<code>... af classfield=is_activated</code>
Return events with uncommon values.	<code>... anomalousvalue action=filter pthresh=0.02</code>
Return results associated with each other (that have at least 3 references to each other).	<code>... associate supcnt=3</code>
For each event, copy the 2nd, 3rd, 4th, and 5th previous values of the 'count' field into the respective fields 'count_p2', 'count_p3', 'count_p4', and 'count_p5'.	<code>... autoregress count p=2-5</code>
Bucket search results into 10 bins, and return the count of raw events for each bucket.	<code>... bucket size bins=10 stats count(_raw) by size</code>
Return the average "thruput" of each "host" for each 5 minute time span.	<code>... bucket _time span=5m stats avg(thruput) by _time host</code>
Return the average (mean) "size" for each distinct "host".	<code>... chart avg(size) by host</code>
Return the the maximum "delay" by "size", where "size" is broken down into a maximum of 10 equal sized buckets.	<code>... chart max(delay) by size bins=10</code>
Return the ratio of the average (mean) "size" to the maximum "delay" for each distinct "host" and "user" pair.	<code>... chart eval(avg(size)/max(delay)) by host user</code>
Return max(delay) for each value of foo split by the value of bar.	<code>... chart max(delay) over foo by bar</code>
Return max(delay) for each value of foo.	<code>... chart max(delay) over foo</code>
Build a contingency table of "datafields" from all events.	<code>... contingency datafield1 datafield2 maxrows=5 maxcols=5 usetotal=F</code>
Calculate the co-occurrence correlation between all fields.	<code>... correlate type=cocur</code>
Return the number of events in the '_internal' index.	<code> eventcount index=_internal</code>
Compute the overall average duration and add 'avgdur' as a new field to each event where the 'duration' field exists	<code>... eventstats avg(duration) as avgdur</code>
Make "_time" continuous with a span of 10 minutes.	<code>... makecontinuous _time span=10m</code>

Remove all outlying numerical values.	... outlier
Return the least common values of the "url" field.	... rare url
Remove duplicates of results with the same "host" value and return the total count of the remaining results.	... stats dc(host)
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
Search the access logs, and return the number of hits from the top 100 values of "referer_domain".	sourcetype=access_combined top limit=100 referer_domain stats sum(count)
For each event, add a count field that represent the number of event seen so far (including that event). i.e., 1 for the first event, 2 for the second, 3, 4 ... and so on	... streamstats count
Graph the average "thruput" of hosts over time.	... timechart span=5m avg(thruput) by host
Create a timechart of average "cpu_seconds" by "host", and remove data (outlying values) that may distort the timechart's axis.	... timechart avg(cpu_seconds) by host outlier action=tf
Calculate the average value of "CPU" each minute for each "host".	... timechart span=1m avg(CPU) by host
Create a timechart of the count of from "web" sources by "host"	... timechart count by host
Compute the product of the average "CPU" and average "MEM" each minute for each "host"	... timechart span=1m eval(avg(CPU) * avg(MEM)) by host
Return the 20 most common values of the "url" field.	... top limit=20 url
Computes a 5 event simple moving average for field 'foo' and write to new field 'smoothed_foo' also computes N=10 exponential moving average for field 'bar' and write to field 'ema10(bar)'.	... trendline sma5(foo) as smoothed_foo ema10(bar)
Reformat the search results.	... timechart avg(delay) by host untable _time host avg_delay
Reformat the search results.	... xyseries delay host_type host

results

append

Append the current results with the tabular results of "fubar".	<code>... chart count by bar append [search fubar chart count by baz]</code>
Joins previous result set with results from 'search foo', on the id field.	<code>... join id [search foo]</code>

filter

Return only anomalous events.	<code>... anomalies</code>
Remove duplicates of results with the same host value.	<code>... dedup host</code>
Combine the values of "foo" with ":" delimiter.	<code>... mvcombine delim=":" foo</code>
Keep only search results whose "_raw" field contains IP addresses in the non-routable class A (10.0.0.0/8).	<code>... regex _raw="(?!\\d)10\\.\\d{1,3}\\\\.\\d{1,3}\\\\.\\d{1,3}(?!\\d) "</code>
Join results with itself on 'id' field.	<code>... selfjoin id</code>
For the current search, keep only unique results.	<code>... uniq</code>
Return "physicsobjs" events with a speed is greater than 100.	<code>sourcetype=physicsobjs where distance/time > 100</code>

generate

All daily time ranges from oct 25 till today	<code> gentimes start=10/25/07</code>
Loads the events that were generated by the search job with id=1233886270.2	<code> loadjob 1233886270.2 events=t</code>
Create new events for each value of multi-value field, "foo".	<code>... mvexpand foo</code>
Run the "mysecurityquery" saved search.	<code> savedsearch mysecurityquery</code>

group

Cluster events together, sort them by their "cluster_count" values, and then return the 20 largest clusters (in data size).	<code>... cluster t=0.9 showcount=true sort - cluster_count head 20</code>
Group search results into 4 clusters based on the values of the "date_hour" and "date_minute" fields.	<code>... kmeans k=4 date_hour date_minute</code>
Group search results that have the same "host" and "cookie", occur within 30 seconds of each other, and do not have a pause greater than 5 seconds between each event into a transaction.	<code>... transaction host cookie maxspan=30s maxpause=5s</code>
Force Splunk to apply event types that you have configured (Splunk Web automatically does this when you view the "eventtype" field).	<code>... typer</code>

order

Return the first 20 results.	<code>... head 20</code>
Reverse the order of a result set.	<code>... reverse</code>
Sort results by "ip" value in ascending order and then by "url" value in descending order.	<code>... sort ip, -url</code>
Return the last 20 results (in reverse order).	<code>... tail 20</code>

read

Display events from the file "messages.1" as if the events were indexed in Splunk.	<code> file /var/log/messages.1</code>
Read in results from the CSV file: "\$SPLUNK_HOME/var/run/splunk/all.csv", keep any that contain the string "error", and save the results to the file: "\$SPLUNK_HOME/var/run/splunk/error.csv"	<code> inputcsv all.csv search error outputcsv errors.csv</code>
Read in "users.csv" lookup file (under \$SPLUNK_HOME/etc/system/lookups or \$SPLUNK_HOME/etc/apps/*/lookups).	<code> inputlookup users.csv</code>

write

Output search results to the CSV file 'mysearch.csv'.	<code>... outputcsv mysearch</code>
Write to "users.csv" lookup file (under \$SPLUNK_HOME/etc/system/lookups or \$SPLUNK_HOME/etc/apps/*/lookups).	<code> outputlookup users.csv</code>

search

external

Run the Python script "myscript" with arguments, myarg1 and myarg2; then, email the results.	<pre>... script python myscript myarg1 myarg2 sendemail to=david@splunk.com</pre>
--	---

search

Keep only search results that have the specified "src" or "dst" values.	<pre>src="10.9.165.*" OR dst="10.9.165.8"</pre>
---	---

subsearch

Get top 2 results and create a search from their host, source and sourcetype, resulting in a single search result with a _query field: _query=("host::mylaptop" AND "source::syslog.log" AND "sourcetype::syslog") OR ("host::bobsllaptop" AND "source::bob-syslog.log" AND "sourcetype::syslog"))	<pre>... head 2 fields source, sourcetype, host format</pre>
Search the time range of each previous result for "failure".	<pre>... localize maxpause=5m map search="search failure starttimeu=\$starttime\$ endtimeu=\$endtime\$"</pre>
Return values of "URL" that contain the string "404" or "303" but not both.	<pre> set diff [search 404 fields url] [search 303 fields url]</pre>

miscellaneous

The iplocation command in this case will never be run on remote peers. All events from remote peers from the initial search for the terms FOO and BAR will be forwarded to the search head where the iplocation command will be run.	<pre>FOO BAR localop iplocation</pre>
--	---

Popular search commands

The following tables lists the more frequently used Splunk search commands. Some of these commands share functions -- you can see a list of these functions with descriptions and examples on the following pages: [Functions for eval and where](#) and [Functions for stats, chart, and timechart](#).

Command	Alias(es)	Description	See also
bucket	bin, discretize	Puts continuous numerical values into discrete sets.	chart , timechart

<code>chart</code>		Returns results in a tabular output for charting. See also, Functions for stats, chart, and timechart .	<code>bucket</code> , <code>sichart</code> , <code>timechart</code>
<code>dedup</code>		Removes subsequent results that match a specified criteria.	<code>uniq</code>
<code>eval</code>		Calculates an expression and puts the value into a field. See also, Functions for eval and where .	<code>where</code>
<code>extract</code>	<code>kv</code>	Extracts field-value pairs from search results.	<code>kvform</code> , <code>multikv</code> , <code>xmlkv</code> , <code>rex</code>
<code>fields</code>		Removes fields from search results.	
<code>head</code>		Returns the first number n of specified results.	<code>reverse</code> , <code>tail</code>
<code>lookup</code>		Explicitly invokes field value lookups.	
<code>multikv</code>		Extracts field-values from table-formatted events.	
<code>rangemap</code>		Sets RANGE field to the name of the ranges that match.	
<code>rare</code>		Displays the least common values of a field.	<code>sirare</code> , <code>stats</code> , <code>top</code>
<code>rename</code>		Renames a specified field; wildcards can be used to specify multiple fields.	
<code>replace</code>		Replaces values of specified fields with a specified new value.	
<code>rex</code>		Specify a Perl regular expression named groups to extract fields while you search.	<code>extract</code> , <code>kvform</code> , <code>multikv</code> , <code>xmlkv</code> , <code>regex</code>
<code>search</code>		Searches Splunk indexes for matching events.	
<code>spath</code>		Extracts key-value pairs from XML or JSON formats.	<code>extract</code> , <code>kvform</code> , <code>multikv</code> , <code>rex</code> , <code>xmlkv</code>
<code>sort</code>		Sorts search results by the specified fields.	<code>reverse</code>
<code>stats</code>		Provides statistics, grouped optionally by fields. See also, Functions for stats, chart, and timechart .	<code>eventstats</code> , <code>top</code> , <code>rare</code>
<code>tail</code>		Returns the last number n of specified results.	<code>head</code> , <code>reverse</code>
<code>timechart</code>		Create a time series chart and corresponding table of statistics. See	<code>chart</code> , <code>bucket</code>

		also, Functions for stats, chart, and timechart .	
<code>top</code>	<code>common</code>	Displays the most common values of a field.	<code>rare</code> , <code>stats</code>
<code>transaction</code>	<code>transam</code>	Groups search results into transactions.	
<code>where</code>		Performs arbitrary filtering on your data. See also, Functions for eval and where .	<code>eval</code>
<code>xmlkv</code>		Extracts XML key-value pairs.	<code>extract</code> , <code>kvform</code> , <code>multikv</code> , <code>rex</code> , <code>spath</code>

Answers

Have questions about search commands? Check out Splunk Answers to see what questions and answers other Splunk users had about the search language.

Splunk for SQL users

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

Concepts

In database terms, Splunk is a distributed, non-relational, semi-structured database with an implicit time dimension. Splunk is not a database in the normative sense -- relational databases require that all table columns be defined up-front and they don't automatically scale by just plugging in new hardware -- but there are analogs to many of the concepts in the database world.

DB Concept	Splunk Concept	Notes
SQL query	Splunk search	A Splunk search retrieves indexed data and can perform transforming and reporting operations. Results from one search can be "piped", or transferred, from command to command, to filter, modify, reorder, and group your results.
table/view	search results	Search results can be thought of as a database view, a dynamically generated table of rows, with columns.

index	index	All values and fields are indexed in Splunk, so there is no need to manually add, update, drop, or even think about indexing columns. Everything can be quickly retrieved automatically.
row	result/event	A result in Splunk is a list of field (i.e., column) values, corresponding to a table row. An event is a result that has a timestamp and raw text. Typically in event is a record from a log file, such as: 173.26.34.223 - - [01/Jul/2009:12:05:27 -0700] "GET /trade/app?action=logout HTTP/1.1" 200 2953
column	field	Fields in Splunk are dynamically returned from a search, meaning that one search might return a set of fields, while another search might return another set. After teaching Splunk how to extract out more fields from the raw underlying data, the same search will return more fields that it previously did. Fields in Splunk are not tied to a datatype.
database/schema	index/app	In Splunk, an 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 app.

From SQL to Splunk

The examples below use the value of the Splunk field "source" as a proxy for "table". In Splunk, "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 table and field names are kept the same from the sql. Also, searches rarely need the FIELDS command to filter out columns as the user interface provides a more convenient method; and you never have to use "AND" in boolean searches, as they are implied between terms.

SQL command	SQL example	Splunk example
SELECT *	SELECT *	source=mytable

	FROM mytable	
WHERE	SELECT * FROM mytable WHERE mycolumn=5	source=mytable mycolumn=5
SELECT	SELECT mycolumn1, mycolumn2 FROM mytable	source=mytable FIELDS mycolumn1, mycolumn2
AND/OR	SELECT * FROM mytable WHERE (mycolumn1="true" OR mycolumn2="red") AND mycolumn3="blue"	source=mytable AND (mycolumn1="true" OR mycolumn2="red") AND mycolumn3="blue"
AS (alias)	SELECT mycolumn AS column_alias FROM mytable	source=mytable RENAME mycolumn as column_alias FIELDS column_alias
BETWEEN	SELECT * FROM mytable WHERE mycolumn BETWEEN 1 AND 5	source=mytable mycolumn<=1 mycolumn>=5
GROUP BY	SELECT mycolumn, avg(mycolumn) FROM mytable WHERE mycolumn=value GROUP BY mycolumn	source=mytable mycolumn=value STATS avg(mycolumn) BY mycolumn FIELDS mycolumn, avg(mycolumn)
HAVING	SELECT mycolumn, avg(mycolumn)	

	FROM mytable WHERE mycolumn=value GROUP BY mycolumn HAVING avg(mycolumn)=value	source=mytable 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=mytable mycolumn="*some text*" <p>Note: The most common search usage in Splunk is actually something that is nearly impossible in SQL -- to search all fields for a substring. The following search will return all rows that contain "some text" anywhere:</p> source=mytable "some text"
ORDER BY	SELECT * FROM mytable ORDER BY mycolumn desc	source=mytable SORT -mycolumn
SELECT DISTINCT	SELECT DISTINCT mycolumn1, mycolumn2 FROM mytable	source=mytable DEDUP mycolumn1 FIELDS mycolumn1, mycolumn2
SELECT TOP	SELECT TOP 5 mycolumn1, mycolumn2	source=mytable

	FROM mytable	TOP mycolumn1, mycolumn2
INNER JOIN	SELECT * FROM mytable1 INNER JOIN mytable2 ON mytable1.mycolumn=mytable2.mycolumn	source=mytable1 JOIN type=inner mycolumn [SEARCH source=mytable2] Note: There are two other methods to do a join: <ul style="list-style-type: none"> • Use the lookup command to add fields from an external table: ... LOOKUP myvaluelookup mycolumn OUTPUT myoutputcolumn <ul style="list-style-type: none"> • Use a subsearch: source=mytable1 [SEARCH source=mytable2 mycolumn2=myvalue FIELDS mycolumn2]
LEFT (OUTER) JOIN	SELECT * FROM mytable1 LEFT JOIN mytable2 ON mytable1.mycolumn=mytable2.mycolumn	source=mytable1 JOIN type=left mycolumn [SEARCH source=mytable2]

SELECT INTO	<pre>SELECT * INTO new_mytable IN mydb2 FROM old_mytable</pre>	<pre>source=old_mytable EVAL source=new_mytable COLLECT index=mydb2</pre> <p>Note: COLLECT is typically used to store expensively calculated fields back into Splunk so that future access is much faster. This current example is atypical but shown for comparison with SQL's command. source will be renamed orig_source</p>
TRUNCATE TABLE	<pre>TRUNCATE TABLE mytable</pre>	<pre>source=mytable DELETE</pre>
INSERT INTO	<pre>INSERT INTO mytable VALUES (value1, value2, value3,...)</pre>	<p>Note: see SELECT INTO. Individual records are not added via the search language, but can be added via the API if need be.</p>
UNION	<pre>SELECT mycolumn FROM mytable1 UNION SELECT mycolumn FROM mytable2</pre>	<pre>source=mytable1 APPEND [SEARCH source=mytable2] DEDUP mycolumn</pre>
UNION ALL	<pre>SELECT * FROM mytable1 UNION ALL</pre>	<pre>source=mytable1 APPEND [SEARCH source=mytable2]</pre>

	SELECT * FROM mytable2	
DELETE	DELETE FROM mytable WHERE mycolumn=5	source=mytable1 mycolumn=5 DELETE
UPDATE	UPDATE mytable SET column1=value, column2=value,... WHERE some_column=some_value	Note: There are a few things to think about when updating records in Splunk. First, you can just add the new values into Splunk (see INSERT INTO) and not worry about deleting the old values, because Splunk 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).

Search Commands and Functions

All search commands

The table below lists all search commands with a short description and links to their individual reference pages. For a quick guide with examples for use of these search commands, refer to the [Search cheat sheet](#).

Some of these commands share functions -- you can see a list of these functions with descriptions and examples on the following pages: [Functions for eval and where](#) and [Functions for stats, chart, and timechart](#).

Command	Alias(es)	Description	See also
abstract	excerpt	Produces a summary of each search result.	highlight
accum		Keeps a running total of the specified numeric field.	autoregress , delta , trendline , streamstats
addcoltotals		Computes an event that contains sum of all numeric fields for previous events.	addtotals , stats
addinfo		Add fields that contain common information about the current search.	search
addtotals		Computes the sum of all numeric fields for each result.	addcoltotals , stats
analyzefields		Analyze numerical fields for their ability to predict another discrete field.	anomalousvalue
anomalies		Computes an "unexpectedness" score for an event.	anomalousvalue , cluster , kmeans , outlier
anomalousvalue		Finds and summarizes irregular, or uncommon, search results.	analyzefields , anomalies , cluster , kmeans , outlier
append		Appends subsearch results to current results.	appendcols , appendcsv , appendlookup , join , set
appendcols		Appends the fields of the subsearch results to current	append , appendcsv , join , set

		results, first results to first result, second to second, etc.	
<code>appendpipe</code>		Appends the result of the subpipeline applied to the current result set to results.	<code>append</code> , <code>appendcols</code> , <code>join</code> , <code>set</code>
<code>associate</code>		Identifies correlations between fields.	<code>correlate</code> , <code>contingency</code>
<code>audit</code>		Returns audit trail information that is stored in the local audit index.	
<code>autoregress</code>		Sets up data for calculating the moving average.	<code>accum</code> , <code>autoregress</code> , <code>delta</code> , <code>trendline</code> , <code>streamstats</code>
<code>bucket</code>	<code>bin</code> , <code>discretize</code>	Puts continuous numerical values into discrete sets.	<code>chart</code> , <code>timechart</code>
<code>bucketdir</code>		Replaces a field value with higher-level grouping, such as replacing filenames with directories.	<code>cluster</code> , <code>dedup</code>
<code>chart</code>		Returns results in a tabular output for charting. See also, Functions for stats, chart, and timechart .	<code>bucket</code> , <code>sichart</code> , <code>timechart</code>
<code>cluster</code>	<code>sic</code>	Clusters similar events together.	<code>anomalies</code> , <code>anomalousvalue</code> , <code>cluster</code> , <code>kmeans</code> , <code>outlier</code>
<code>collect</code>	<code>stash</code>	Puts search results into a summary index.	<code>overlap</code>
<code>concurrency</code>		Uses a duration field to find the number of "concurrent" events for each event.	<code>timechart</code>
<code>contingency</code>	<code>counttable</code> , <code>ctable</code>	Builds a contingency table for two fields.	<code>associate</code> , <code>correlate</code>
<code>convert</code>		Converts field values into numerical values.	<code>eval</code>
<code>correlate</code>		Calculates the correlation between different fields.	<code>associate</code> , <code>contingency</code>
<code>crawl</code>		Crawls the filesystem for new sources to index.	
<code>dbinspect</code>		Returns information about the specified index.	
<code>dedup</code>			<code>uniq</code>

		Removes subsequent results that match a specified criteria.	
<code>delete</code>		Delete specific events or search results.	
<code>delta</code>		Computes the difference in field value between nearby results.	<code>accum</code> , <code>autoregress</code> , <code>trendline</code> , <code>streamstats</code>
<code>diff</code>		Returns the difference between two search results.	
<code>dispatch</code>		Encapsulates long running, streaming reports.	
<code>erex</code>		Allows you to specify example or counter example values to automatically extract fields that have similar values.	<code>extract</code> , <code>kvform</code> , <code>multikv</code> , <code>regex</code> , <code>rex</code> , <code>xmlkv</code>
<code>eval</code>		Calculates an expression and puts the value into a field. See also, Functions for eval and where .	<code>where</code>
<code>eventstats</code>		Adds summary statistics to all search results.	<code>stats</code>
<code>extract</code>	<code>kv</code>	Extracts field-value pairs from search results.	<code>kvform</code> , <code>multikv</code> , <code>xmlkv</code> , <code>rex</code>
<code>fieldformat</code>		Expresses how to render a field at output time without changing the underlying value.	<code>eval</code> , <code>where</code>
<code>fields</code>		Removes fields from search results.	
<code>file</code>		This command is no longer supported. See inputcsv .	
<code>filldown</code>		Replaces NULL values with the last non-NULL value.	<code>fillnull</code>
<code>fillnull</code>		Replaces null values with a specified value.	
<code>format</code>		Takes the results of a subsearch and formats them into a single result.	
<code>gauge</code>		Transforms results into a format suitable for display by the Gauge chart types.	
<code>gentimes</code>		Generates time-range results.	
<code>head</code>			<code>reverse</code> , <code>tail</code>

		Returns the first number n of specified results.	
<code>highlight</code>		Causes Splunk Web to highlight specified terms.	
<code>history</code>		Returns a history of searches formatted as an events list or as a table.	<code>search</code>
<code>iconify</code>		Causes Splunk Web to make a unique icon for each value of the fields listed.	<code>highlight</code>
<code>input</code>		Adds sources to Splunk or disables sources from being processed by Splunk.	
<code>inputcsv</code>		Loads search results from the specified CSV file.	<code>loadjob</code> , <code>outputcsv</code>
<code>inputlookup</code>		Loads search results from a specified static lookup table.	<code>inputcsv</code> , <code>join</code> , <code>lookup</code> , <code>outputlookup</code>
<code>iplocation</code>		Extracts location information from IP addresses.	
<code>join</code>		SQL-like joining of results from the main results pipeline with the results from the subpipeline.	<code>selfjoin</code> , <code>appendcols</code>
<code>kmeans</code>		Performs k-means clustering on selected fields.	<code>anomalies</code> , <code>anomalousvalue</code> , <code>cluster</code> , <code>outlier</code>
<code>kvform</code>		Extracts values from search results, using a form template.	<code>extract</code> , <code>kvform</code> , <code>multikv</code> , <code>xmlkv</code> , <code>rex</code>
<code>loadjob</code>		Loads search results from a specified CSV file.	<code>inputcsv</code>
<code>localize</code>		Returns a list of the time ranges in which the search results were found.	<code>map</code> , <code>transaction</code>
<code>lookup</code>		Explicitly invokes field value lookups.	
<code>makecontinuous</code>		Makes a field that is supposed to be the x-axis continuous (invoked by <code>chart</code> / <code>timechart</code>)	<code>chart</code> , <code>timechart</code>
<code>makemv</code>		Change a specified field into a multivalued field during a search.	<code>mvcombine</code> , <code>mvexpand</code> , <code>nomv</code>
<code>map</code>		A looping operator, performs a search over each search result.	

<code>metadata</code>		Returns a list of source, sourcetypes, or hosts from a specified index or distributed search peer.	<code>dbinspect</code>
<code>metasearch</code>		Retrieves event metadata from indexes based on terms in the logical expression.	<code>metadata, search</code>
<code>multikv</code>		Extracts field-values from table-formatted events.	
<code>mvcombine</code>		Combines events in search results that have a single differing field value into one result with a multivalue field of the differing field.	<code>mvexpand, makemv, nomv</code>
<code>mvexpand</code>		Expands the values of a multivalue field into separate events for each value of the multivalue field.	<code>mvcombine, makemv, nomv</code>
<code>nomv</code>		Changes a specified multivalued field into a single-value field at search time.	<code>makemv, mvcombine, mvexpand</code>
<code>outlier</code>	<code>outlierfilter</code>	Removes outlying numerical values.	<code>anomalies, anomalousvalue, cluster, kmeans</code>
<code>outputcsv</code>		Outputs search results to a specified CSV file.	<code>inputcsv, outputtext</code>
<code>outputlookup</code>		Writes search results to the specified static lookup table.	<code>inputlookup, lookup, outputcsv, outputlookup</code>
<code>outputtext</code>		Outputs the raw text field (<code>_raw</code>) of results into the <code>_xml</code> field.	<code>outputtext</code>
<code>overlap</code>		Finds events in a summary index that overlap in time or have missed events.	<code>collect</code>
<code>predict</code>		Enables you to use time series algorithms to predict future values of fields.	<code>x11</code>
<code>rangemap</code>		Sets RANGE field to the name of the ranges that match.	
<code>rare</code>		Displays the least common values of a field.	<code>sirare, stats, top</code>

<code>regex</code>		Removes results that do not match the specified regular expression.	<code>rex, search</code>
<code>relevancy</code>		Calculates how well the event matches the query.	
<code>reltime</code>		Converts the difference between 'now' and '_time' to a human-readable value and adds this value to the field, 'reltime', in your search results.	<code>convert</code>
<code>rename</code>		Renames a specified field; wildcards can be used to specify multiple fields.	
<code>replace</code>		Replaces values of specified fields with a specified new value.	
<code>rest</code>		Access a REST endpoint and display the returned entities as search results.	
<code>reverse</code>		Reverses the order of the results.	<code>head, sort, tail</code>
<code>rex</code>		Specify a Perl regular expression named groups to extract fields while you search.	<code>extract, kvform, multikv, xmlkv, regex</code>
<code>rtorder</code>		Buffers events from real-time search to emit them in ascending time order when possible.	
<code>run</code>		See <code>script</code> .	
<code>savedsearch</code>	<code>macro, savedsplunk</code>	Returns the search results of a saved search.	
<code>script</code>	<code>run</code>	Runs an external Perl or Python script as part of your search.	
<code>scrub</code>		Anonymizes the search results.	
<code>search</code>		Searches Splunk indexes for matching events.	
<code>searchtxn</code>		Finds transaction events within specified search constraints.	<code>transaction</code>
<code>selfjoin</code>		Joins results with itself.	<code>join</code>
<code>sendemail</code>		Emails search results to a specified email address.	

<code>set</code>		Performs set operations on subsearches.	
<code>setfields</code>		Sets the field values for all results to a common value.	<code>eval</code> , <code>fillnull</code> , <code>rename</code>
<code>sichart</code>		Summary indexing version of <code>chart</code> .	<code>chart</code> , <code>sitimechart</code> , <code>timechart</code>
<code>sirare</code>		Summary indexing version of <code>rare</code> .	<code>rare</code>
<code>sistats</code>		Summary indexing version of <code>stats</code> .	<code>stats</code>
<code>sitimechart</code>		Summary indexing version of <code>timechart</code> .	<code>chart</code> , <code>sichart</code> , <code>timechart</code>
<code>sitop</code>		Summary indexing version of <code>top</code> .	<code>top</code>
<code>sort</code>		Sorts search results by the specified fields.	<code>reverse</code>
<code>spath</code>		Provides a straightforward means for extracting fields from structured data formats, XML and JSON.	<code>xpath</code>
<code>stats</code>		Provides statistics, grouped optionally by fields. See also, Functions for stats, chart, and timechart .	<code>eventstats</code> , <code>top</code> , <code>rare</code>
<code>strcat</code>		Concatenates string values.	
<code>streamstats</code>		Adds summary statistics to all search results in a streaming manner.	<code>eventstats</code> , <code>stats</code>
<code>table</code>		Creates a table using the specified fields.	<code>fields</code>
<code>tags</code>		Annotates specified fields in your search results with tags.	<code>eval</code>
<code>tail</code>		Returns the last number <code>n</code> of specified results.	<code>head</code> , <code>reverse</code>
<code>timechart</code>		Create a time series chart and corresponding table of statistics. See also, Functions for stats, chart, and timechart .	<code>chart</code> , <code>bucket</code>
<code>top</code>	<code>common</code>	Displays the most common values of a field.	<code>rare</code> , <code>stats</code>

<code>transaction</code>	<code>transam</code>	Groups search results into transactions.	
<code>transpose</code>		Reformats rows of search results as columns.	
<code>trendline</code>		Computes moving averages of fields.	<code>timechart</code>
<code>typeahead</code>		Returns typeahead information on a specified prefix.	
<code>typelearner</code>		Generates suggested eventtypes.	<code>typer</code>
<code>typer</code>		Calculates the eventtypes for the search results.	<code>typelearner</code>
<code>uniq</code>		Removes any search that is an exact duplicate with a previous result.	<code>dedup</code>
<code>untable</code>		Converts results from a tabular format to a format similar to <code>stats</code> output. Inverse of <code>xyseries</code> and <code>maketable</code> .	
<code>where</code>		Performs arbitrary filtering on your data. See also, Functions for eval and where .	<code>eval</code>
<code>x11</code>		Enables you to determine the trend in your data by removing the seasonal pattern.	<code>predict</code>
<code>xmlkv</code>		Extracts XML key-value pairs.	<code>extract</code> , <code>kvform</code> , <code>multikv</code> , <code>rex</code>
<code>xmlunescape</code>		Unescapes XML.	
<code>xpath</code>		Redefines the XML path.	
<code>xyseries</code>		Converts results into a format suitable for graphing.	

Functions for eval and where

These are functions that you can use with the `eval` and `where` commands and as part of eval expressions.

Function	Description	Example(s)
<code>abs (X)</code>	This function takes a number X and returns its absolute	This example returns the <code>absnum</code> , whose values are the absolute value of the <code>number</code> field:

	value.	<code>... eval absnum=abs(number)</code>
<code>case(X, "Y", ...)</code>	This function takes pairs of arguments X and Y. X arguments are Boolean expressions that, when evaluated to TRUE, return the corresponding Y argument. The function defaults to NULL if none are true.	This example returns descriptions for the corresponding http <code>... eval description=case(error == 404, "N 500, "Internal Server Error", error == 200,</code>
<code>ceil(X), ceiling(X)</code>	This function returns the ceiling of a number X.	This example returns n=2: <code>... eval n=ceil(1.9)</code>
<code>cidrmatch("X", Y)</code>	This function identifies IP addresses that belong to a particular subnet. The function uses two arguments: the first is the CIDR subnet, which is contained in quotes; the second is the IP address to match, which may be values in a field.	This example returns a field, <code>addy</code> , whose values are the field <code>ip</code> that match the subnet: <code>... eval addy=cidrmatch("123.132.32.0/25",</code>
<code>coalesce(X, ...)</code>	This function takes an arbitrary number of arguments and returns the first value that is not null.	Let's say you have a set of events where the IP address is <code>clientip</code> or <code>ipaddress</code> . This example defines a new field <code>ip</code> with a value of either <code>clientip</code> or <code>ipaddress</code> if <code>clientip</code> is not NULL (exists in that event): <code>... eval ip=coalesce(clientip, ipaddress)</code>
<code>commands(X)</code>	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. (This is generally not recommended for use except for analysis of audit.log events.)	<code>... eval x=commands("search foo stats co</code> returns a multivalue field x, that contains 'search',
<code>exact(X)</code>	This function evaluates an expression X using double precision floating point arithmetic.	<code>... eval n=exact(3.14 * num)</code>
<code>exp(X)</code>	This function takes a number X and returns e^X .	This example returns $y=e^3$: <code>... eval y=exp(3)</code>
<code>floor(X)</code>	This function returns the floor of a number X.	This example returns 1:

		... eval n=floor(1.9)
if (X,Y,Z)	This function takes three arguments. The first argument X is a Boolean expression. If X evaluates to TRUE, the result is the second argument Y. Optionally, if X evaluates to FALSE, the result evaluates to the third argument Z.	This example looks at the values of error and returns err=0 if error is 200 or less, otherwise returns err=Error: ... eval err=if(error == 200, "OK", "Error")
isbool (X)	This function takes one argument X and returns TRUE if X is Boolean.	... eval n=if(isbool(field),"yes","no") or ... where isbool(field)
isint (X)	This function takes one argument X and returns TRUE if X is an integer.	... eval n=isint(field) or ... where isint(field)
isnotnull (X)	This function takes one argument X and returns TRUE if X is not NULL. This is a useful check for whether or not a field (X) contains a value.	... eval n=if(isnotnull(field),"yes","no") or ... where isnotnull(field)
isnull (X)	This function takes one argument X and returns TRUE if X is NULL.	... eval n=if(isnull(field),"yes","no") or ... where isnull(field)
isnum (X)	This function takes one argument X and returns TRUE if X is a number.	... eval n=if(isnum(field),"yes","no") or ... where isnum(field)
isstr ()	This function takes one argument X and returns TRUE if X is a string.	... eval n=if(isstr(field),"yes","no") or ... where isstr(field)
len (X)	This function returns the character length of a string X.	... eval n=len(field)
like (X,"Y")		This example returns islike=TRUE if the field value starts with Y

	This function takes two arguments, a field X and a quoted string Y, and returns TRUE if and only if the first argument is like the SQLite pattern in Y.	... eval islike=like(field, "foo%") or ... where like(field, "foo%")
ln(X)	This function takes a number X and returns its natural log.	This example returns the natural log of the values of bytes: ... eval lnBytes=ln(bytes)
log(X, Y)	This function takes either one or two numeric arguments and returns the log of the first argument X using the second argument Y as the base. If the second argument Y is omitted, this function evaluates the log of number X with base 10.	... eval num=log(number, 2)
lower(X)	This function takes one string argument and returns the lowercase version. The upper() function also exists for returning the uppercase version.	This example returns the value provided by the field username: ... eval username=lower(username)
ltrim(X, Y)	This function takes one or two string 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 trimmed.	This example returns x="abcZZ": ... eval x=ltrim(" ZZZZabcZZ ", " Z")
match(X, Y)	This function compares the regex string Y to the value of X and returns a Boolean value; it returns T (true) if X matches the pattern defined by Y.	This example returns true IF AND ONLY IF field matches the address. Note that the example uses ^ and \$ to perform a full match: ... eval n=match(field, "^\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\$")
max(X, ...)	This function takes an arbitrary number of arguments X, that is numbers or strings, and returns the max; strings are greater than numbers.	This example returns either "foo" or field, depending on the value of field: ... eval n=max(1, 3, 6, 7, "foo", field)
md5(X)	This function computes and returns the MD5 hash of a string value X.	... eval n=md5(field)
min(X, ...)		This example returns 1:

	This function takes an arbitrary number of arguments X, that is numbers or strings, and returns the min; strings are greater than numbers.	... eval n=min(1, 3, 6, 7, "foo", field)
mvappend(X, "Y", Z)	This function takes three arguments, fields X and Z and a quoted string Y, and returns a multivalued result. The value of Y and the values of the field Z are appended to the values of field X. The fields X and Z can be either multi or single valued fields.	
mvcount(X)	This function takes an field X and returns the number of values of that field if the field is multivalued, 1 if the field is single valued, and NULL otherwise.	... eval n=mvcount(multifield)
mvfilter(X)	<p>This function filters a multi-valued field based on an arbitrary Boolean expression X. The Boolean expression X can reference ONLY ONE field at a time.</p> <p>Note:This function will return NULL values of the field x as well. If you don't want the NULL values, use the expression: mvfilter(x!=NULL).</p>	<p>This example returns all values of the field email that end in</p> <p>... eval n=mvfilter(match(email, "\.net\$") "\.org\$"))</p>
mvfind(X, "Y")	Appears in 4.2.2. This function tries to find a value in multivalued field X that matches the regular expression Y. If a match exists, the index of the first matching value is returned (beginning with zero). If no values match, NULL is returned.	... eval n=mvfind(mymvfield, "err\d+")
mvindex(X, Y, Z)		Since indexes start at zero, this example returns the third va

	<p>This function takes two or three arguments, field X and numbers Y and Z, and returns a subset of the multivalued field using the indexes provided.</p> <p>For <code>mvindex(mvfield, startindex, [endindex])</code>, <code>endindex</code> is inclusive and optional; both <code>startindex</code> and <code>endindex</code> can be negative, where -1 is the last element. If <code>endindex</code> is not specified, it returns just the value at <code>startindex</code>. If the indexes are out of range or invalid, the result is NULL.</p>	<pre>... eval n=mvindex(multifield, 2)</pre>
<code>mvjoin(X, Y)</code>	<p>This function takes two arguments, multi-valued field X and string delimiter Y, and joins the individual values of X using Y.</p>	<p>This example joins together the individual values of "foo" using the delimiter:</p> <pre>... eval n=mvjoin(foo, ";")</pre>
<code>mvrangle(X, Y, Z)</code>	<p>This function creates a multivalue field for a range of numbers. It takes up to three arguments: a starting number X, an ending number Y (exclusive), and an optional step increment Z. If the increment is a timespan (such as '7d'), the starting and ending numbers are treated as epoch times.</p>	<p>This example returns a multivalue field with the values 1, 3, 5, 7, 9, 11:</p> <pre>... eval mv=mvrangle(1,11,2)</pre>
<code>mvzip(X, Y)</code>	<p>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, etc. Similar to Python's zip command.</p>	<pre>... eval n=server=mvzip(hosts,ports)</pre>
<code>now()</code>		

	This function takes no arguments and returns the time that the search was started. The time is represented in Unix time or seconds since epoch.	
<code>null()</code>	This function takes no arguments and returns NULL. The evaluation engine uses NULL to represent "no value"; setting a field to NULL clears its value.	
<code>nullif(X,Y)</code>	This function takes two arguments, fields X and Y, and returns the X if the arguments are different. It returns NULL, otherwise.	<code>... eval n=nullif(fieldA,fieldB)</code>
<code>pi()</code>	This function takes no arguments and returns the constant pi to 11 digits of precision.	
<code>pow(X,Y)</code>	This function takes two numeric arguments X and Y and returns X^Y .	
<code>random()</code>	This function takes no arguments and returns a pseudo-random number ranging from zero to $2^{31}-1$, for example: 0...2147483647	
<code>relative_time(X,Y)</code>	This function takes an epochtime time, X, as the first argument and a relative time specifier, Y, as the second argument and returns the epochtime value of Y applied to X.	<code>... eval n=relative_time(now(), "-1d@d")</code>
<code>replace(X,Y,Z)</code>	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.	This example returns date with the month and day numbers 1/12/2009 the return value would be 12/1/2009: <code>... eval n=replace(date, "^(\d{1,2})/(\d{1,2})/(\d{4})", "\$2/\$1/\$3")</code>
<code>round(X,Y)</code>	This function takes one or two numeric arguments X	This example returns n=4:

	and Y, returning X rounded to the amount of decimal places specified by Y. The default is to round to an integer.	<pre>... eval n=round(3.5)</pre> <p>This example returns n=2.56:</p> <pre>... eval n=round(2.555, 2)</pre>
<code>rtrim(X, Y)</code>	This function takes one or two string 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 trimmed.	<p>This example returns n="ZZZZabc":</p> <pre>... eval n=rtrim(" ZZZZabcZZ ", " Z")</pre>
<code>searchmatch(X)</code>	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.	<pre>... eval n=searchmatch("foo AND bar")</pre>
<code>sigfig(X)</code>	This function takes one argument X, a number, and rounds that number to the appropriate number of significant figures.	<p>1.00*1111 = 1111, but</p> <pre>... eval n=sigfig(1.00*1111)</pre> <p>returns n=1110.</p>
<code>spath(X, Y)</code>	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. If Y is a literal string, it needs quotes, <code>spath(X, "Y")</code> . If Y is a field name (with values that are the location paths), it doesn't need quotes. This may result in a multivalued field. Read more about the spath search command.	<p>This example returns the values of locDesc elements:</p> <pre>... eval locDesc=spath(_raw, "vendorProductSet.product.desc.locDesc")</pre> <p>This example returns the hashtags from a twitter event:</p> <pre>eval output=spath(_raw, "entities.hashtags")</pre>
<code>split(X, "Y")</code>	This function takes two arguments, field X and delimiting character Y. It splits the value(s) of X on the delimiter Y and returns X as a multi-valued field.	<pre>... eval n=split(foo, ";")</pre>

<code>sqrt (X)</code>	This function takes one numeric argument X and returns its square root.	This example returns 3: ... eval n=sqrt(9)
<code>strftime (X, Y)</code>	This function takes an epochtime value, X, as the first argument and renders it as a string using the format specified by Y. For a list and descriptions of format options, refer to the topic "Common time format variables" .	This example returns the hour and minute from the <code>_time</code> field: ... eval n=strftime(_time, "%H:%M")
<code>strptime (X, Y)</code>	This function takes a time represented by a string, X, and parses it using the format specified by Y. For a list and descriptions of format options, refer to the topic "Common time format variables" .	This example returns the hour and minute from the <code>timeStr</code> field: ... eval n=strptime(timeStr, "%H:%M")
<code>substr (X, Y, Z)</code>	<p>This function takes either two or three arguments, where X is a string and Y and Z are numeric. It returns a substring of X, starting at the index specified by Y with the number of characters specified by Z. If Z is not given, it returns the rest of the string.</p> <p>The indexes follow SQLite semantics; they start at 1. Negative indexes can be used to indicate a start from the end of the string.</p>	This example concatenates "str" and "ing" together, returning "string": ... eval n=substr("string", 1, 3) + substr("ing", 1, 3)
<code>time ()</code>	This function returns the wall-clock time with microsecond resolution. The value of <code>time()</code> will be different for each event based on when that event was processed by the <code>eval</code> command.	
<code>tonumber ("X", Y)</code>		This example returns "164": ... eval n=tonumber("164", 10)

	<p>This function converts the input string X to a number, where Y is optional and used to define the base of the number to convert to. Y can be 2..36, and defaults to 10. If it cannot parse the input to a number, the function returns NULL.</p>	<pre>... eval n=tonumber("0A4",16)</pre>
<p>tostring(X,Y)</p>	<p>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".</p> <p>This function requires at least one argument X; if X is a number, the second argument Y is optional and can be "hex" "commas" or "duration":</p> <ul style="list-style-type: none"> • <code>tostring(X,"hex")</code> converts X to hexadecimal. • <code>tostring(X,"commas")</code> formats X with commas and, if the number includes decimals, rounds to nearest two decimal places. • <code>tostring(X,"duration")</code> converts seconds X to readable time format HH:MM:SS. 	<p>This example returns "True 0xF 12,345.68":</p> <pre>... eval n=tostring(1==1) + " " + tostring(12345.6789, "commas")</pre> <p>This example returns <code>foo=615</code> and <code>foo2=00:10:15</code>:</p> <pre>eval foo = tostring(foo, "duration")</pre>
<p>trim(X,Y)</p>	<p>This function takes one or two string arguments X and Y and returns X with the characters in Y trimmed from</p>	<p>This example returns "abc":</p> <pre>... eval n=trim(" ZZZZabcZZ ", " Z")</pre>

	both sides. If Y is not specified, spaces and tabs are trimmed.	
<code>typeof(X)</code>	This function takes one argument and returns a string representation of its type.	This example returns "NumberStringBoolInvalid": ... eval n=typeof(12) + typeof("string") + typeof(badfield)
<code>upper(X)</code>	This function takes one string argument and returns the uppercase version. The <code>lower()</code> function also exists for returning the lowercase version.	This example returns the value provided by the field username: ... eval n=upper(username)
<code>urldecode(X)</code>	This function takes one URL string argument X and returns the unescaped or decoded URL string.	This example returns "http://www.splunk.com/download?r=l": ... eval n=urldecode("http%3A%2F%2Fwww.splunk.com%2Fdownload?r=l")
<code>validate(X,Y,...)</code>	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 and defaults to NULL if all are True.	This example runs a simple check for valid ports: ... eval n=validate(isint(port), "ERROR: Port is not an integer", port >= 1 AND port <= 65535, "ERROR: Port is out of range")

Functions for stats, chart, and timechart

These are statistical functions that you can use with the `chart`, `stats`, and `timechart` commands.

- Functions that are relevant for stats are also relevant for `eventstats` and `streamstats`.
- Functions that are relevant for chart, stats, and timechart are also relevant for their respective summary indexing counterparts: `sichart`, `sistats`, and `sitimechart`.
- Functions that are relevant for sparklines will say as much. Note that sparklines apply only to `chart` and `stats`.

Function	Description	Command(s)	Example(s)
<code>avg(X)</code>	This function returns the average of the values of field X. See	<code>chart</code> , <code>stats</code> , <code>timechart</code> , <code>sparkline()</code>	This examples returns the average response time:

	also, mean(X).		avg(responseTime)
c(X) count(X)	This function returns the number of occurrences of the field X. To indicate a specific field value to match, format X as eval(field="value").	chart, stats, timechart, sparkline()	<p>This example returns the count of events where status has the value "404":</p> <pre>count(eval(status="404"))</pre> <p>These generate sparklines for the counts of events. The first looks at the _raw field. The second counts events with a user field:</p> <pre>sparkline(count)</pre> <pre>sparkline(count(user))</pre>
dc(X) distinct_count(X)	This function returns the count of distinct values of the field X.	chart, stats, timechart, sparkline()	<p>This example generates sparklines for the distinct count of devices and renames the field, "numdevices":</p> <pre>sparkline(dc(device)) AS numdevices</pre> <p>This example counts the distinct sources for each sourcetype, and buckets the count for each five minute spans:</p> <pre>sparkline(dc(source,5m)) by sourcetype</pre>
earliest(X)	This function returns the chronologically earliest seen occurrence of a value of a field X.	chart, stats, timechart	
estdc(X)	This function returns the estimated count of the distinct values of the field X.	chart, stats, timechart	
estdc_error(X)	This function returns the theoretical error of the estimated count of the distinct values of	chart, stats, timechart	

	the field X. The error represents a ratio of $\text{abs}(\text{estimate_value} - \text{real_value})/\text{real_value}$.		
<code>first(X)</code>	This function returns the first seen value of the field X. In general, the first seen value of the field the most recent instance of this field, relative to the input order of events into the stats command.	<code>chart, stats, timechart</code>	
<code>last(X)</code>	This function returns the last seen value of the field X. In general, the last seen value of the field relative to the input order of events into the stats command.	<code>chart, stats, timechart</code>	
<code>latest(X)</code>	This function returns the chronologically latest seen occurrence of a value of a field X.	<code>chart, stats, timechart</code>	
<code>list(X)</code>	This function returns the list of all values of the field X as a multi-value entry. The order of the values reflects the order of input events.	<code>chart, stats, timechart</code>	
<code>max(X)</code>	This function returns the maximum value of the field X. If the values of X are non-numeric, the max is found from lexicographic ordering.	<code>chart, stats, timechart, sparkline()</code>	This example returns the maximum value of "size": <code>max(size)</code>
<code>mean(X)</code>	This function returns the arithmetic mean of the field X. See also, <code>avg(X)</code> .	<code>chart, stats, timechart, sparkline()</code>	This example returns the mean of "kbps" values: <code>mean(kbps)</code>
<code>median(X)</code>	This function returns the middle-most value of the field X.	<code>chart, stats, timechart</code>	
<code>min(X)</code>			

	This function returns the minimum value of the field X. If the values of X are non-numeric, the min is found from lexicographic ordering.	<code>chart, stats, timechart</code>	
<code>mode (X)</code>	This function returns the most frequent value of the field X.	<code>chart, stats, timechart</code>	
<code>p<X> (Y) perc<X> (Y) exactperc<X> (Y) upperperc<X> (Y)</code>	This function returns the X-th percentile value of the field Y. The functions <code>perc</code> , <code>p</code> , and <code>upperperc</code> give approximate values for the integer percentile requested. The approximation algorithm we use provides a strict bound of the actual value at for any percentile. The functions <code>perc</code> and <code>p</code> return a single number that represents the lower end of that range while <code>upperperc</code> gives the approximate upper bound. <code>exactperc</code> provides the exact value, but will be very expensive for high cardinality fields.	<code>chart, stats, timechart</code>	This example returns the 5th percentile value of a field "total": <code>perc5 (total)</code>
<code>per_day (X)</code>	This function returns the values of field X per day.	<code>timechart</code>	This example returns the values of "total" per day. <code>per_day (total)</code>
<code>per_hour (X)</code>	This function returns the values of field X per hour.	<code>timechart</code>	This example returns the values of "total" per hour. <code>per_hour (total)</code>
<code>per_minute (X)</code>	This function returns the values of field X per minute.	<code>timechart</code>	This example returns the values of "total" per minute. <code>per_minute (total)</code>
<code>per_second (X)</code>	This function returns the values of field X per second.	<code>timechart</code>	This example returns values of "kb" per second:

			<code>per_second(kb)</code>
<code>range(X)</code>	This function returns the difference between the max and min values of the field X ONLY IF the value of X are numeric.	<code>chart, stats, timechart, sparkline()</code>	
<code>stdev(X)</code>	This function returns the sample standard deviation of the field X.	<code>chart, stats, timechart, sparkline()</code>	This example returns the standard deviation of wildcarded fields "*delay" which can apply to both, "delay" and "xdelay". <code>stdev(*delay)</code>
<code>stdevp(X)</code>	This function returns the population standard deviation of the field X.	<code>chart, stats, timechart, sparkline()</code>	
<code>sum(X)</code>	This function returns the sum of the values of the field X.	<code>chart, stats, timechart, sparkline()</code>	<code>sum(eval(date_hour * date_minute))</code>
<code>sumsq(X)</code>	This function returns the sum of the squares of the values of the field X.	<code>chart, stats, timechart, sparkline()</code>	
<code>values(X)</code>	This function returns the list of all distinct values of the field X as a multi-value entry. The order of the values is lexicographical.	<code>chart, stats, timechart</code>	
<code>var(X)</code>	This function returns the sample variance of the field X.	<code>chart, stats, timechart, sparkline()</code>	
<code>varp(X)</code>	This function returns the population variance of the field X.	<code>chart, stats, timechart, sparkline().</code>	

Common date and time format variables

This topic lists the variables that are used to define time formats in the `eval` functions `strftime()` and `strptime()` and for describing timestamps in event data.

Time variables

Variable	Description
%Ez	Splunk specific, timezone in minutes.
%H	Hour (24-hour clock) as a decimal number, includes leading zeros. (00 to 23)
%I	Hour (12-hour clock), includes leading zeros. (01-12)
%k	Like %H, the hour (24-hour clock) as a decimal number; but a leading zero is replaced by a space. (0 to 23)
%M	Minute as a decimal number. (00 to 59)
%N	Subseconds with width. (%3N = milliseconds, %6N = microseconds, %9N = nanoseconds)
%p	AM or PM.
%Q	The subsecond component of 1970-01-01 00:00:00 UTC. (%3Q = milliseconds, %6Q = microseconds, %9Q = nanoseconds with values of 000-999)
%S	Second as a decimal number. (00 to 61)
%s	The Unix Epoch Time timestamp, or the number of seconds since the Epoch: 1970-01-01 00:00:00 +0000 (UTC). (1352395800 is Thu Nov 8 09:30:00 2012)
%T	The time in 24-hour notation (%H:%M:%S).
%Z	The timezone abbreviation. (EST for Eastern Time)
%:z	The timezone offset from UTC, in hour and minute: +hhmm or -hhmm. (-0500 for Eastern Time)
%%	A literal "%" character.

Date variables

Variable	Description
%F	Equivalent to %Y-%m-%d (the ISO 8601 date format).

Specifying days

Variable	Description
%A	Full weekday name. (Sunday, ..., Saturday)
%a	Abbreviated weekday name. (Sun, ... ,Sat)
%d	Day of the month as a decimal number, includes a leading zero. (01 to 31)
%e	Like %d, the day of the month as a decimal number, but a leading zero is replaced by a space. (1 to 31)
%j	Day of year as a decimal number, includes a leading zero. (001 to 366)

%w	Weekday as a decimal number. (0 = Sunday, ..., 6 = Saturday)
----	--

Specifying months

Variable	Description
%b	Abbreviated month name. (Jan, Feb, etc.)
%B	Full month name. (January, February, etc.)
%m	Month as a decimal number. (01 to 12)

Specifying year

Variable	Description
%y	Year as a decimal number, without the century. (00 to 99)
%Y	Year as a decimal number with century. (2012)

Examples

Time format string	Result
%Y-%m-%d	2012-12-31
%y-%m-%d	12-12-31
%b %d, %Y	Feb 11, 2008
q %d%b '%y = %Y-%m-%d	q 23 Apr '12 = 2012-04-23

Time modifiers for search

You can use time modifiers to customize the time range of a search by specifying a time to start or stop, or change the format of the timestamps in the search results.

List of time modifiers

We recommend using the `earliest` and/or `latest` modifiers to specify custom and relative time ranges. Also, when specifying relative time, you can use `now` to refer to the current time.

Modifier	Syntax	Description
<code>earliest</code>	<code>earliest=[+ -]<time_integer><time_unit>@<time_unit></code>	Specify the earliest time for the time range of your search.
<code>latest</code>	<code>latest=[+ -]<time_integer><time_unit>@<time_unit></code>	

		Specify the latest time for the time range of your search.
now	<code>now()</code>	Refers to the current time. If set to earliest, <code>now()</code> is the start of the search.
time	<code>time()</code>	In real-time searches, <code>time()</code> is the current machine time.

For more information about customizing your search window, see "Specify real-time time range windows in your search" in the Search manual.

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) and, optionally, a "snap to" time unit:

```
[+|-]<time_integer><time_unit>@<time_unit>.
```

1. Begin your string with a plus (+) or minus (-) to indicate the offset from the current time.
2. Define your time amount with a number and a unit; the supported time units are:

- second: s, sec, secs, second, seconds
- minute: m, min, minute, minutes
- hour: h, hr, hrs, hour, hours
- day: d, day, days
- week: w, week, weeks
- days of the week: w0 (Sunday), w1, w2, w3, w4, w5 and w6 (Saturday)
- month: mon, month, months
- quarter: q, qtr, qtrs, quarter, quarters
- year: y, yr, yrs, year, years

Note: For Sunday, you can specify w0 and w7.

For example, to start your search an hour ago, use either

earliest=-h

or,

earliest=-60m

When specifying single time amounts, the number one is implied; 's' is the same as '1s', 'm' is the same as '1m', 'h' is the same as '1h', etc.

3. If you want, specify a "snap to" time unit; this 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 in Step 2. For example, @w, @week, and @w0 for Sunday; @month for the beginning of the month; and @q, @qtr, or @quarter for the beginning of the most recent quarter (Jan 1, Apr 1, Jul 1, or Oct 1).
- 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 (12AM) and subtract 2 hours from that time.
- When snapping to the nearest or latest time, Splunk 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 don't specify a time offset before the "snap to" amount, Splunk 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 AM.

Example 1: To search events from the beginning of the current week:

earliest=@w0

Example 2: To search events from the last full business week:

earliest=-7d@w1 latest=@w6

Example 3: To search with an exact date as boundary, such as from November 5th at 8PM to November 12 at 8PM, use the timeformat: %m/%d/%Y:%H:%M:%S

earliest="5/11/2012:20:00:00" latest="12/11/2012:20:00:00"

More time modifiers

These search time modifiers are still valid, BUT **may be removed and their function no longer supported in a future release.**

Modifier	Syntax	Description
daysago	daysago=<int>	Search events within the last integer number of days.
enddaysago	enddaysago=<int>	Set an end time for an integer number of days before now.
endhoursago	endhoursago=<int>	Set an end time for an integer number of hours before now.
endminutesago	endminutesago=<int>	Set an end time for an integer number of minutes before now.
endmonthsago	endmonthsago=<int>	Set an end time for an integer number of months before now.
endtime	endtime=<string>	Search for events before the specified time (exclusive of the specified time). Use timeformat to specify how the timestamp is formatted.
endtimeu	endtimeu=<int>	Search for events before the specific epoch time (Unix time). .
hoursago	hoursago=<int>	Search events within the last integer number of hours.
minutesago	minutesago=<int>	Search events within the last integer number of minutes.
monthsago	monthsago=<int>	Search events within the last integer number of months.
<searchtimespandays	searchtimespandays=<int>	Search within a specified range of days (expressed as an integer).
searchtimespanhours	searchtimespanhours=<int>	Search within a specified range of hours (expressed as an integer).
searchtimespanminutes	searchtimespanminutes=<int>	Search within a specified range of minutes (expressed as an integer).
searchtimespanmonths	searchtimespanmonths=<int>	Search within a specified range of months (expressed as an integer).
startdaysago	startdaysago=<int>	Search the specified number of days before the present time.
starthoursago	starthoursago=<int>	Search the specified number of hours before the present time.

startminutesago	startminutesago=<int>	Search the specified number of minutes before the present time.
startmonthsago	startmonthsago=<int>	Search the specified number of months before the present time.
starttime	starttime=<timestamp>	Search from the specified date and time to the present (inclusive of the specified time).
starttimeu	starttimeu=<int>	Search from the specific epoch (Unix time).
timeformat	timeformat=<string>	Set the timeformat for the starttime and endtime modifiers. By default: timeformat=%m/%d/%Y:%H:%M:%S

List of data types

This topic is out of date.

This page lists the data types used to define the syntax of the search language. Learn more about the commands used in these examples by referring to the [search command reference](#).

after-opt

Syntax: timeafter=<int>(s|m|h|d)?

Description: the amount of time to add to endtime (ie expand the time region forward in time)

anovalue-action-option

Syntax: action=(annotate|filter|summary)

Description: If action is ANNOTATE, a new field is added to the event containing the anomalous value that indicates the anomaly score of the value. If action is FILTER, events with anomalous value(s) are isolated. If action is SUMMARY, a table summarizing the anomaly statistics for each field is generated.

anovalue-pthresh-option

Syntax: pthresh=<num>

Description: Probability threshold (as a decimal) that has to be met for a value to be deemed anomalous

associate-improv-option

Syntax: improv=<num>

Description: Minimum entropy improvement for target key. That is, entropy(target key) - entropy(target key given reference key/value) must be greater than or equal to this.

associate-option

Syntax:

<[associate-supcnt-option](#)>|<[associate-supfreq-option](#)>|<[associate-improv-option](#)>

Description: Associate command options

associate-supcnt-option

Syntax: supcnt=<int>

Description: Minimum number of times the reference key=reference value combination must be appear. Must be a non-negative integer.

associate-supfreq-option

Syntax: supfreq=<num>

Description: Minimum frequency of reference key=reference value combination, as a fraction of the number of total events.

before-opt

Syntax: timebefore=<int>(s|m|h|d)?

Description: the amount of time to subtract from starttime (ie expand the time region backwards in time)

bucket-bins

Syntax: bins=<int>

Description: Sets the maximum number of bins to discretize into. Given this upper-bound guidance, the bins will snap to human sensible bounds.

Example: bins=10

bucket-span

Syntax: span=(<span-length>|<log-span>)

Description: Sets the size of each bucket.

Example: span=2d

Example: span=5m

Example: span=10

bucket-start-end

Syntax: (start=|end=)<num>

Description: Sets the minimum and maximum extents for numerical buckets.

bucketing-option

Syntax: <bucket-bins>|<bucket-span>|<bucket-start-end>

Description: Discretization option.

by-clause

Syntax: by <field-list>

Description: Fields to group by.

Example: BY addr, port

Example: BY host

cmp

Syntax: =|!=|<|<=|>|>=

Description: None

collapse-opt

Syntax: collapse=<bool>

Description: whether to collapse terms that are a prefix of another term and the event count is the same

Example: collapse=f

collect-addinfo

Syntax: No syntax

Description: None

collect-addtime

Syntax: addtime=<bool>

Description: whether to prefix a time into each event if the event does not contain a `_raw` field. The first found field of the following times is used: `info_min_time`, `_time`, `now()` defaults to true

collect-arg

Syntax: <collect-addtime> | <collect-index> | <collect-file> | <collect-spool> | <collect-marker> | <collect-testmode>

Description: None

collect-file

Syntax: file=<string>

Description: name of the file where to write the events to. Optional, default "`<random-num>_events.stash`". The following placeholders can be used in the file name `$timestamp$`, `$random$` and will be replaced with a timestamp, a random number respectively

collect-index

Syntax: index=<string>

Description: name of the index where splunk should add the events to. Note: the index must exist for events to be added to it, the index is NOT created automatically.

collect-marker

Syntax: marker=<string>

Description: a string, usually of key-value pairs, to append to each event written out. Optional, default ""

collect-spool

Syntax: spool=<bool>

Description: If set to true (default is true), the summary indexing file will be written to Splunk's spool directory, where it will be indexed automatically. If set to false, file will be written to `$SPLUNK_HOME/var/run/splunk`.

collect-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. (defaults to false)

comparison-expression

Syntax: <field><cmp><value>

Description: None

connected-opt

Syntax: connected=<bool>

Description: Relevant iff fields is not empty. Controls whether an event that is not inconsistent and not consistent with the fields of a transaction, opens a new transaction (connected=t) or is added to the transaction. An event can be not inconsistent and not consistent if it contains fields required by the transaction but none of these fields has been instantiated in the transaction (by a previous event addition).

contingency-maxopts

Syntax: (maxrows|maxcols)=<int>

Description: Maximum number of rows or columns. If the number of distinct values of the field exceeds this maximum, the least common values will be ignored. A value of 0 means unlimited rows or columns.

contingency-mincover

Syntax: (mincolcover|minrowcover)=<num>

Description: Cover only this percentage of values for the row or column field. If the number of entries needed to cover the required percentage of values exceeds maxrows or maxcols, maxrows or maxcols takes precedence.

contingency-option

Syntax:

<contingency-maxopts>|<contingency-mincover>|<contingency-usetotal>|<contingency-tota

Description: Options for the contingency table

contingency-totalstr

Syntax: totalstr=<field>

Description: Field name for the totals row/column

contingency-usetotal

Syntax: usetotal=<bool>

Description: Add row and column totals

convert-auto

Syntax: auto("(" (<wc-field>)? ")")?

Description: Automatically convert the field(s) 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.

Example: ... | convert auto(*delay) as *delay_secs

Example: ... | convert auto(*) as *_num

Example: ... | convert auto(delay) auto(xdelay)

Example: ... | convert auto(delay) as delay_secs

Example: ... | convert auto

Example: ... | convert auto()

Example: ... | convert auto(*)

convert-ctime

Syntax: ctime("("<wc-field>?")")

Description: Convert an epoch time to an ascii human readable time. Use timeformat option to specify exact format to convert to.

Example: ... | convert timeformat="%H:%M:%S" ctime(_time) as timestr

convert-dur2sec

Syntax: dur2sec("("<wc-field>?")")

Description: Convert a duration format "D+HH:MM:SS" to seconds.

Example: ... | convert dur2sec(*delay)

Example: ... | convert dur2sec(xdelay)

convert-function

Syntax:

<convert-auto>|<convert-dur2sec>|<convert-mstime>|<convert-memk>|<convert-none>|<co

Description: None

convert-memk

Syntax: memk("(" <wc-field>? ")")

Description: Convert a {KB, MB, GB} denominated size quantity into a KB

Example: ... | convert memk(VIRT)

convert-mktime

Syntax: mktime("(" <wc-field>? ")")

Description: Convert an human readable time string to an epoch time. Use timeformat option to specify exact format to convert from.

Example: ... | convert mktime(timestr)

convert-mstime

Syntax: mstime("(" <wc-field>? ")")

Description: Convert a MM:SS.SSS format to seconds.

convert-none

Syntax: none("(" <wc-field>? ")")

Description: In the presence of other wildcards, indicates that the matching fields should not be converted.

Example: ... | convert auto(*) none(foo)

convert-num

Syntax: num("(" <wc-field>? ")")

Description: Like auto(), except non-convertible values are removed.

convert-rmcomma

Syntax: rmcomma("(" <wc-field>? ")")

Description: Removes all commas from value, e.g. '1,000,000.00' -> '1000000.00'

convert-rmunit

Syntax: rmunit("(" <wc-field>? ")")

Description: Looks for numbers at the beginning of the value and removes trailing text.

Example: ... | convert rmunit(duration)

copyresults-dest-option

Syntax: dest=<string>

Description: The destination file where to copy the results to. The string is interpreted as path relative to SPLUNK_HOME and (1) should point to a .csv file and (2) the file should be located either in etc/system/lookups/ or etc/apps/<app-name>/lookups/

copyresults-sid-option

Syntax: sid=<string>

Description: The search id of the job whose results are to be copied. Note, the user who is running this command should have permission to the job pointed by this id.

correlate-type

Syntax: type=cocur

Description: Type of correlation to calculate. Only available option currently is the co-occurrence matrix, which contains the percentage of times that two fields exist in the same events.

count-opt

Syntax: count=<int>

Description: The maximum number of results to return

Example: count=10

crawl-option

Syntax: <string>=<string>

Description: Override settings from crawl.conf.

Example: root=/home/bob

daysago

Syntax: daysago=<int>

Description: Search the last N days. (equivalent to startdaysago)

debug-method

Syntax: optimize|roll|logchange|validate|delete|sync|sleep|rescan

Description: The available commands for debug command

dedup-consecutive

Syntax: consecutive=<bool>

Description: Only eliminate events that are consecutive

dedup-keepempty

Syntax: keepempty=<bool>

Description: If an event contains a null value for one or more of the specified fields, the event is either retained (if keepempty=true) or discarded

dedup-keepevents

Syntax: keepevents=<bool>

Description: Keep all events, remove specific values instead

default

Syntax: No syntax

Description: None

delim-opt

Syntax: delim=<string>

Description: A string used to delimit the original event values in the transaction event fields.

email_address

Syntax: <string>

Description: None

Example: bob@smith.com

email_list

Syntax: <email_address> (, <email_address>)*

Description: None

Example: "bob@smith.com, elvis@presley.com"

end-opt

Syntax: endswith=<transam-filter-string>

Description: A search or eval filtering expression which if satisfied by an event marks the end of a transaction

Example: endswith=eval(speed_field > max_speed_field/12)

Example: endswith=(username=foobar)

Example: endswith=eval(speed_field > max_speed_field)

Example: endswith="logout"

enddaysago

Syntax: enddaysago=<int>

Description: A short cut to set the end time. endtime = now - (N days)

endhoursago

Syntax: endhoursago=<int>

Description: A short cut to set the end time. endtime = now - (N hours)

endminutesago

Syntax: endminutesago=<int>

Description: A short cut to set the end time. endtime = now - (N minutes)

endmonthsago

Syntax: endmonthsago=<int>

Description: A short cut to set the start time. starttime = now - (N months)

endtime

Syntax: endtime=<string>

Description: All events must be earlier or equal to this time.

endtimeu

Syntax: endtime=<int>

Description: Set the end time to N seconds since the epoch. (unix time)

erex-examples

Syntax: ""<string>(, <string>)*""

Description: None

Example: "foo, bar"

eval-bool-exp

Syntax: (NOT|!)? (<eval-compare-exp>|<eval-function-call>)
((AND|OR|XOR) <eval-expression>)*

Description: None

eval-compare-exp

Syntax: (<field>|<string>|<num>) (<|>|<=|>=|!=|==|LIKE)
<eval-expression>

Description: None

eval-concat-exp

Syntax: ((<field>|<string>|<num>) (.
<eval-expression>)*)|((<field>|<string>) (+ <eval-expression>)*)

Description: concatenate fields and strings

Example: first_name." ".last_nameSearch

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. The following are the basic operations you can perform with eval. 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 may not produce valid results if the values are not numerical. For addition, Splunk can concatenate the two operands if they are both strings. When concatenating values with '.', Splunk treats both values as strings regardless of their actual type.

eval-field

Syntax: <field>

Description: A field name for your evaluated value.

Example: velocity

eval-function

Syntax:

abs|case|cidrmatch|coalesce|exact|exp|floor|if|ifnull|isbool|isint|isnotnull|isnull|isnum|isstr|len

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}\$")oo", field)

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")

eval-function-call

Syntax: <eval-function> "(" <eval-expression> ("," <eval-expression>)* ")"

Description: None

eval-math-exp

Syntax: (<field>|<num>) ((+|-|*|/|%> <eval-expression>)*

Description: None

Example: pi() * pow(radius_a, 2) + pi() * pow(radius_b, 2)

eval-field

Syntax: "eval("<eval-expression>")"

Description: A dynamically eval'd field

event-id

Syntax: <int>:<int>

Description: a splunk internal event id

eventtype-specifier

Syntax: eventtype=<string>

Description: Search for events that match the specified eventtype

eventtypetag-specifier

Syntax: eventtypetag=<string>

Description: Search for events that would match all eventtypes tagged by the string

extract-opt

Syntax:

(segment=<bool>)|(auto=<bool>)|(reload=<bool>)|(limit=<int>)|(maxchars=<int>)|(mv_add=

Description: Extraction options. "segment" specifies whether to note the locations of key/value pairs with the results (internal, false). "auto" specifies whether to perform automatic '=' based extraction (true). "reload"

specifies whether to force reloading of props.conf and transforms.conf (false). "limit" specifies how many automatic key/value pairs to extract (50). "kvdelim" string specifying a list of character delimiters that separate the key from the value "pairedelim" string specifying a list of character delimiters that separate the key-value pairs from each other "maxchars" specifies how many characters to look into the event (10240). "mv_add" whether to create multivalued fields. Overrides MV_ADD from transforms.conf "clean_keys" whether to clean keys. Overrides CLEAN_KEYS from transforms.conf

Example: reload=true

Example: auto=false

extractor-name

Syntax: <string>

Description: A stanza that can be found in transforms.conf

Example: access-extractions

fields-opt

Syntax: fields=<string>? (,<string>)*

Description: DEPRECATED: The preferred usage of transaction is for list of fields to be specified directly as arguments. E.g. 'transaction foo bar' rather than 'transaction fields="foo,bar"' The 'fields' constraint takes a list of fields. For search results to be members of a transaction, for each field specified, if they have a value, it must have the same value as other members in that transaction. For example, a search result that has host=mylaptop can never be in the same transaction as a search result that has host=myserver, if host is one of the constraints. A search result that does not have a host value, however, can be in a transaction with another search result that has host=mylaptop, because they are not inconsistent.

Example: fields=host,cookie

grouping-field

Syntax: <field>

Description: By default, the typelearner initially groups events by the value of the grouping-field, and then further unifies and merges those groups, based on the keywords they contain. The default grouping field is "punct" (the punctuation seen in _raw).

Example: host

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 initially group events

Example: maxlen=30

host-specifier

Syntax: host=<string>

Description: Search for events from the specified host

hosttag-specifier

Syntax: hosttag=<string>

Description: Search for events that have hosts that are tagged by the string

hoursago

Syntax: hoursago=<int>

Description: Search the last N hours. (equivalent to starthoursago)

increment

Syntax: <int:increment>(s|m|h|d)?

Description: None

Example: 1h

index-expression

Syntax: \"<string>\"|<term>|<search-modifier>

Description: None

index-specifier

Syntax: index=<string>

Description: Search the specified index instead of the default index

input-option

Syntax: <string>=<string>

Description: Override settings from inputs.conf.

Example: root=/home/bob

join-options

Syntax: usetime=<bool> | earlier=<bool> | overwrite=<bool> | max=<int>

Description: Options to the join command. usetime indicates whether to limit matches to sub results that are earlier or later (depending on the 'earlier' option which is only valid when usetime=true) than the main result to join with, default = false. 'overwrite' indicates if fields from the sub results should overwrite those from the main result if they have the same field name (default = true). max indicates the maximum number of sub results each main result can join with. (default = 1, 0 means no limit).

Example: max=3

Example: usetime=t earlier=f

Example: overwrite=f

Example: usetime=t

keepevicted-opt

Syntax: keepevicted=<bool>

Description: Whether to output evicted transactions. Evicted transactions can be distinguished from non-evicted transactions by checking the value of the 'evicted' field, which is set to '1' for evicted transactions

key-list

Syntax: (<string>)*

Description: a list of keys that are ANDed to provide a filter for surrounding command

kmeans-cnumfield

Syntax: cfield=<field>

Description: Controls the field name for the cluster number for each event

kmeans-distype

Syntax: dt=(l1norm|l2norm|cityblock|sqeuclidean|cosine)

Description: Distance metric to use (L1/L1NORM equivalent to CITYBLOCK). L2NORM equivalent to SQUEUCLIDEAN

kmeans-iters

Syntax: maxiters=<int>

Description: Maximum number of iterations allowed before failing to converge

kmeans-k

Syntax: k=<int>(-<int>)?

Description: Number of initial clusters to use. Can be a range, in which case each value in the range will be used once and summary data given.

kmeans-options

Syntax:

<kmeans-reps>|<kmeans-iters>|<kmeans-tol>|<kmeans-k>|<kmeans-cnumfield>|<kmeans-

Description: Options for kmeans command

kmeans-reps

Syntax: reps=<int>

Description: Number of times to repeat kmeans using random starting clusters

kmeans-showlabel

Syntax: showlabel=<bool>

Description: Controls whether or not the cluster number is added to the data.

kmeans-tol

Syntax: tol=<num>

Description: Algorithm convergence tolerance

lit-value

Syntax: <string>|<num>

Description: None

lmaxpause-opt

Syntax: maxpause=<int>(s|m|h|d)?

Description: the maximum (inclusive) time between two consecutive events in a contiguous time region

log-span

Syntax: (<num>)?log(<num>)?

Description: Sets to log based span, first number is coefficient, second number is base coefficient, if supplied, must be real number ≥ 1.0 and $<$ base base, if supplied, must be real number > 1.0 (strictly greater than 1)

Example: 2log5

Example: log

logical-expression

Syntax: (NOT)?

<logical-expression>)|<comparison-expression>|(<logical-expression>

OR? <logical-expression>)

Description: None

max-time-opt

Syntax: max_time=<int>

Description: None

Example: max_time=3

maxevents-opt

Syntax: maxevents=<int>

Description: The maximum number of events in a transaction. If the value is negative this constraint is disabled.

maxinputs-opt

Syntax: maxinputs=<int>

Description: Determines how many of the top results are passed to the script.

Example: maxinputs=1000

maxopenevents-opt

Syntax: maxopenevents=<int>

Description: Specifies the maximum number of events (which are) part of open transactions before transaction eviction starts happening, using LRU policy.

maxopentxn-opt

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.

maxpause-opt

Syntax: maxpause=<int>(s|m|h|d)?

Description: The maxpause constraint requires there be no pause between a transaction's events of greater than maxpause. If value is negative, disable the maxpause constraint.

maxsearchesoption

Syntax: maxsearches=<int>

Description: The maximum number of searches to run. Will generate warning if there are more search results.

Example: maxsearches=42

maxspan-opt

Syntax: maxspan=<int>(s|m|h|d)?

Description: The maxspan constraint requires the transaction's events to span less than maxspan. If value is negative, disable the maxspan constraint.

memcontrol-opt

Syntax: <maxopentxn-opt> | <maxopenevents-opt> | <keepevicted-opt>

Description: None

metadata-delete-restrict

Syntax: (host::|source::|sourcetype::)<string>

Description: restrict the deletion to the specified host, source or sourcetype.

metadata-type

Syntax: hosts|sources|sourcetypes

Description: controls which metadata type that will be returned

minutesago

Syntax: minutesago=<int>

Description: Search the last N minutes. (equivalent to startminutesago)

monthsago

Syntax: monthsago=<int>

Description: Search the last N months. (equivalent to startmonthsago)

multikv-copyattrs

Syntax: copyattrs=<bool>

Description: Controls the copying of non-metadata attributes from the original event to extract events (default = true)

multikv-fields

Syntax: fields <field-list>

Description: Filters out from the extracted events fields that are not in the given field list

multikv-filter

Syntax: filter <field-list>

Description: If specified, a table row must contain one of the terms in the list before it is extracted into an event

multikv-forceheader

Syntax: forceheader=<int>

Description: Forces the use of the given line number (1 based) as the table's header. By default a header line is searched for.

multikv-multitable

Syntax: multitable=<bool>

Description: Controls whether or not there can be multiple tables in a single _raw in the original events? (default = true)

multikv-noheader

Syntax: noheader=<bool>

Description: Allow tables with no header? If no header fields would be named column1, column2, ... (default = false)

multikv-option

Syntax:

<multikv-copyattrs>|<multikv-fields>|<multikv-filter>|<multikv-forceheader>|<multikv-multitable>

Description: Multikv available options

multikv-rmorig

Syntax: rmorig=<bool>

Description: Controls the removal of original events from the result set (default=true)

mvlist-opt

Syntax: mvlist=<bool>|<field-list>

Description: Flag controlling whether the multivalued fields of the transaction are (1) a list of the original events ordered in arrival order or (2) a set of unique field values ordered lexicographically. If a comma/space delimited list of fields is provided only those fields are rendered as lists

outlier-action-opt

Syntax: action=(remove|transform)

Description: What to do with outliers. RM | REMOVE removes the event containing the outlying numerical value. TF | TRANSFORM truncates the outlying value to the threshold for outliers and prefixes the value with "000"

outlier-option

Syntax:

<outlier-type-opt>|<outlier-action-opt>|<outlier-param-opt>|<outlier-uselower-opt>

Description: Outlier options

outlier-param-opt

Syntax: param=<num>

Description: Parameter controlling the threshold of outlier detection. For type=IQR, an outlier is defined as a numerical value that is outside of param multiplied the inter-quartile range.

outlier-type-opt

Syntax: type=iqr

Description: Type of outlier detection. Only current option is IQR (inter-quartile range)

outlier-uselower-opt

Syntax: uselower=<bool>

Description: Controls whether to look for outliers for values below the median

prefix-opt

Syntax: prefix=<string>

Description: The prefix to do typeahead on

Example: prefix=source

quoted-str

Syntax: "" <string> ""

Description: None

readlevel-int

Syntax: 0|1|2|3

Description: How deep to read the events, 0 : just source/host/sourcetype, 1 : 0 with _raw, 2 : 1 with kv, 3 2 with types (deprecated in 3.2)

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)"

rendering-opt

Syntax: <delim-opt> | <mvlist-opt>

Description: None

result-event-opt

Syntax: events=<bool>

Description: Option controlling whether to load the events or results of a job. (default: false)

Example: events=t

savedsearch-identifier

Syntax:

savedsearch="<user-string>:<application-string>:<search-name-string>"

Description: The unique identifier of a savedsearch whose artifacts need to be loaded. A savedsearch is uniquely identified by the triplet {user, application, savedsearch name}.

Example: savedsearch="admin:search:my saved search"

savedsearch-macro-opt

Syntax: nosubstitution=<bool>

Description: If true, no macro replacements are made.

savedsearch-opt

Syntax: <savedsearch-macro-opt>|<savedsearch-replacement-opt>

Description: None

savedsearch-replacement-opt

Syntax: <string>=<string>

Description: A key value pair to be used in macro replacement.

savedsplunk-specifier

Syntax: (savedsearch|savedsplunk)=<string>

Description: Search for events that would be found by specified search/splunk

savedsplunkoption

Syntax: <string>

Description: Name of saved search

Example: mysavedsearch

script-arg

Syntax: <string>

Description: An argument passed to the script.

Example: to=bob@mycompany.com

script-name-arg

Syntax: <string>

Description: The name of the script to execute, minus the path and file extension.

Example: sendemail

search-modifier

Syntax:

<source-type-specifier>|<host-specifier>|<source-specifier>|<savedsplunk-specifier>|<eventtype-specifier>

Description: None

searchoption

Syntax: search=\"<string>\"

Description: Search to run map on

Example: search="search starttimeu::\$start\$ endtimeu::\$end\$"

searchtimespandays

Syntax: searchtimespandays=<int>

Description: None

searchtimespanhours

Syntax: searchtimespanhours=<int>

Description: The time span operators are always applied from the last time boundary set. Therefore, if an endtime operator is closest to the left of a timespan operator, it will be applied to the starttime. If you had 'enddaysago::1 searchtimespanhours::5', it would be equivalent to 'starthoursago::29 enddaysago::1'.

searchtimespanminutes

Syntax: searchtimespanminutes=<int>

Description: None

searchtimespanmonths

Syntax: searchtimespanmonths=<int>

Description: None

select-arg

Syntax: <string>

Description: Any value sql select arguments, per the syntax found at http://www.sqlite.org/lang_select.html. If no "from results" is specified in the select-arg it will be inserted it automatically. Runs a SQL Select query against passed in search results. All fields referenced in the select statement must be prefixed with an underscore. Therefore, "ip" should be references as "_ip" and "_raw" should be referenced as "__raw". Before the select command is executed, the previous search results are put into a temporary database table called "results". If a row has no values, "select" ignores it to prevent blank search results.

selfjoin-options

Syntax: overwrite=<bool> | max=<int> | keepsingle=<int>

Description: The selfjoin joins each result with other results that have the same value for the join fields. 'overwrite' controls if fields from these 'other' results should overwrite fields of the result used as the basis for the join (default=true). max indicates the maximum number of 'other' results each main result can join with. (default = 1, 0 means no limit). 'keepsingle' controls whether or not results with a unique value for the join fields (and thus no other results to join with) should be retained. (default = false)

Example: max=3

Example: keepsingle=t

Example: overwrite=f

server-list

Syntax: (<string>)*

Description: A list of possibly wildcarded servers changes in the context of the differences. Try it see if it makes sense. * - header=[true | false] : optionally you can show a header that tries to explain the diff output * - attribute=[attribute name] : you can choose to diff just a single attribute of the results.

sid-opt

Syntax: <string>

Description: The search id of the job whose artifacts need to be loaded.

Example: 1233886270.2

single-agg

Syntax: count|<stats-func>(<field>)

Description: A single aggregation applied to a single field (can be evaled 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

slc-option

Syntax:

(t=<num>|<delims=<string>|<showcount=<bool>|<countfield=<field>|<labelfield=<field>|<field

Description: Options for configuring the simple log clusters. "T=" sets the threshold which must be > 0.0 and < 1.0. The closer the threshold is to 1, the more similar events have to be in order to be considered in the same cluster. Default is 0.8 "delims" 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" if yes, this shows the size of each cluster (default = true unless labelonly is set to true) "countfield" name of field to write cluster size to, default = "cluster_count" "labelfield" name of field to write cluster number to, default = "cluster_label" "field" name of field to

analyze, default = `_raw` "labelonly" if true, instead of reducing each cluster to a single event, keeps all original events and merely labels with them their cluster number "match" determines the similarity method used, defaulting to `termlist`. `termlist` requires the exact same ordering of terms, `termset` allows for an unordered set of terms, and `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'

Example: `t=0.9 delims=" ;:" showcount=true countfield="SLCCNT" labelfield="LABEL" field=_raw labelonly=true`

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`

sort-field

Syntax: `<field> | ((auto|str|ip|num) "(" <field> ")")`

Description: a sort field may be a field or a sort-type and field. sort-type can be "ip" to interpret the field's values as ip addresses. "num" to treat them as numbers, "str" to order lexicographically, and "auto" to make the determination automatically. If no type is specified, it is assumed to be "auto"

Example: `host`

Example: `_time`

Example: `ip(source_addr)`

Example: `str(pid)`

Example: `auto(size)`

source-specifier

Syntax: `source=<string>`

Description: Search for events from the specified source

sourcetype-specifier

Syntax: `sourcetype=<string>`

Description: Search for events from the specified sourcetype

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

Example: 5m

Example: 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.

srcfields

Syntax: (<field>|<quoted-str>)* (<field>|<quoted-str>)* (<field>|<quoted-str>)*

Description: Fields should either be key names or quoted literals

start-opt

Syntax: startswith=<transam-filter-string>

Description: A search or eval filtering expression which if satisfied by an event marks the beginning of a new transaction

Example: startswith=eval(speed_field < max_speed_field/12)

Example: startswith=(username=foobar)

Example: startswith=eval(speed_field < max_speed_field)

Example: startswith="login"

startdaysago

Syntax: startdaysago=<int>

Description: A short cut to set the start time. starttime = now - (N days)

starthoursago

Syntax: starthoursago=<int>

Description: A short cut to set the start time. starttime = now - (N hours)

startminutesago

Syntax: startminutesago=<int>

Description: A short cut to set the start time. starttime = now - (N minutes)

startmonthsago

Syntax: startmonthsago=<int>

Description: A short cut to set the start time. starttime = now - (N months)

starttime

Syntax: starttime=<string>

Description: Events must be later or equal to this time. Must match time format.

starttimeu

Syntax: starttimeu=<int>

Description: Set the start time to N seconds since the epoch. (unix time)

stats-agg

Syntax: <stats-func>("(" (<evaluated-field> | <wc-field>)? ")")?

Description: A specifier formed by a aggregation function applied to a field or set of fields. As of 4.0, it can also be an aggregation function applied to a 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"})

Example: max(size)

Example: stdev(*delay)

Example: 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

Example: avg(kbps)

stats-c

Syntax: count

Description: The count of the occurrences of the field.

stats-dc

Syntax: distinct-count

Description: The count of distinct values of the field.

stats-first

Syntax: first

Description: The first seen value of the field.

stats-func

Syntax:

<stats-c>|<stats-dc>|<stats-mean>|<stats-stdev>|<stats-var>|<stats-sum>|<stats-min>|<stats-max>

Description: Statistical aggregators.

stats-last

Syntax: last

Description: The last seen value of the field.

stats-list

Syntax: list

Description: List of all values of this field as a multi-value entry. Order of values reflects order of input events.

stats-max

Syntax: max

Description: The maximum value of the field (lexicographic, if non-numeric).

stats-mean

Syntax: avg

Description: The arithmetic mean of the field.

stats-median

Syntax: median

Description: The middle-most value of the field.

stats-min

Syntax: min

Description: The minimum value of the field (lexicographic, if non-numeric).

stats-mode

Syntax: mode

Description: The most frequent value of the field.

stats-perc

Syntax: perc<int>

Description: The n-th percentile value of this field.

stats-range

Syntax: range

Description: The difference between max and min (only if numeric)

stats-stdev

Syntax: stdev|stdevp

Description: The {sample, population} standard deviation of the field.

stats-sum

Syntax: sum

Description: The sum of the values of the field.

stats-values

Syntax: values

Description: List of all distinct values of this field as a multi-value entry.
Order of values is lexicographical.

stats-var

Syntax: var|varp

Description: The {sample, population} variance of the field.

subsearch

Syntax: [<string>]

Description: Specifies a subsearch.

Example: [search 404 | select url]

subsearch-options

Syntax: maxtime=<int> | maxout=<int> | timeout=<int>

Description: controls how the subsearch is executed.

tc-option

Syntax:

<bucketing-option>|(usenull=<bool>)|(useother=<bool>)|(nullstr=<string>)|(otherstr=<string>)

Description: Options for controlling the behavior of splitting by a field. In addition to the bucketing-option: 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

time-modifier

Syntax:

<starttime>|<startdaysago>|<startminutesago>|<starthoursago>|<startmonthsago>|<starttim

Description: None

time-opts

Syntax: (<timeformat>)? (<time-modifier>)*

Description: None

timeformat

Syntax: timeformat=<string>

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

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

timescale

Syntax: <ts-sec>|<ts-min>|<ts-hr>|<ts-day>|<ts-month>|<ts-subseconds>

Description: Time scale units.

timestamp

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

Description: None

Example: 10/1/07:12:34:56

Example: -5

top-opt

Syntax:

(showcount=<bool>)|(showperc=<bool>)|(rare=<bool>)|(limit=<int>)|(countfield=<string>)|(percentfield=<string>)

Description: Top arguments: showcount: Whether to create a field called "count" (see countfield option) with the count of that tuple. (T) showperc: Whether to create a field called "percent" (see percentfield option) with the relative prevalence of that tuple. (T) rare: When set and calling as top or common, evokes the behavior of calling as rare. (F) limit: Specifies how many tuples to return, 0 returns all values. (10) countfield: Name of new field to write count to (default is "count") percentfield: Name of new field to write percentage to (default is "percent")

transaction-name

Syntax: <string>

Description: The name of a transaction definition from transactions.conf to be used for finding transactions. If other arguments (e.g., maxspan) are provided as arguments to transam, they overrule the value specified in the transaction definition.

Example: purchase_transaction

transam-filter-string

Syntax: "<search-expression>" | (<quoted-search-expression>) |
eval(<eval-expression>)

Description: Where: \i\ <search-expression> is a valid search expression that does not contain quotes\i\ <quoted-search-expression> is a valid search expression that contains quotes\i\ <eval-expression> is a valid eval expression that evaluates to a boolean

Example: eval(distance/time < max_speed)

Example: "user=mildred"

Example: ("search literal")

Example: (name="foo bar")

trend_type

Syntax: (sma|ema|wma)<num>

Description: The type of trend to compute which consist of a trend type and trend period (integer between 2 and 10000)

Example: sma10

ts-day

Syntax: days

Description: Time scale in days.

ts-hr

Syntax: hours

Description: Time scale in hours.

ts-min

Syntax: minutes

Description: Time scale in minutes.

ts-month

Syntax: months

Description: Time scale in months.

ts-sec

Syntax: seconds

Description: Time scale in seconds.

ts-subseconds

Syntax: us|ms|cs|ds

Description: Time scale in microseconds("us"), milliseconds("ms"), centiseconds("cs"), or deciseconds("ds")

txn_definition-opt

Syntax: <maxspan-opt> | <maxpause-opt> | <maxevents-opt> | <field-list> | <start-opt> | <end-opt> | <connected-opt>

Description: None

value

Syntax: <lit-value>|<field>

Description: None

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

where-comp

Syntax: <wherein-comp>|<wherethresh-comp>

Description: A criteria for the where clause.

wherein-comp

Syntax: (in|notin) (top|bottom)<int>

Description: A where-clause criteria that requires the aggregated series value be in or not in some top or bottom grouping.

Example: notin top2

Example: in bottom10

Example: in top5

wherethresh-comp

Syntax: (<|>)()?<num>

Description: A where-clause criteria that requires the aggregated series value be greater than or less than some numeric threshold.

Example: < 100

Example: > 2.5

x-field

Syntax: <field>

Description: Field to be used as the x-axis

y-data-field

Syntax: <field>

Description: Field that contains the data to be charted

y-name-field

Syntax: <field>

Description: Field that contains the values to be used as data series labels

Search Command Reference

abstract

Synopsis

Produces a summary of each search result.

Syntax

abstract [maxterms=*int*] [maxlines=*int*]

Optional arguments

maxterms

Syntax: maxterms=<int>

Description: The maximum number of terms to match.

maxlines

Syntax: maxlines=<int>

Description: The maximum number of lines to match.

Description

This data processing command produces an abstract (summary) of each search result. The importance of a line in being in the summary is scored by how many search terms it contains as well as how many search terms are on nearby lines. If a line has a search term, its neighboring lines also partially match, and may be returned to provide context. When there are jumps between the lines selected, lines are prefixed with an ellipsis (...).

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

Synopsis

Keeps a running total of a specified numeric field.

Syntax

```
accum <field> [AS <newfield>]
```

Required arguments

field

Syntax: <string>

Description: The name of a field with numeric values.

Optional arguments

newfield

Syntax: <string>

Description: The name of a field to write the results to.

Description

For each event where *field* is a number, keep a running total of the sum of this number and write it out to either the same field, or a **newfield** if specified.

Examples

Example 1: Save the running total of "count" in a field called "total_count".

```
... | accum count AS total_count
```

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

Synopsis

Computes a new event with fields that represent the sum of all numeric fields in previous events.

Syntax

```
addcoltotals [labelfield=<field>] [label=<string>]
```

Optional arguments

`label`

Syntax: `label=<string>`

Description: If `labelfield` is specified, it will be added to this summary event with the value set by the 'label' option.

`labelfield`

Syntax: `labelfield=<field>`

Description: Specify a name for the summary event.

Description

The `addcoltotals` command 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.

Examples

Example 1: Compute the sums of all the fields, and put the sums in a summary event called "change_name".

```
... | addcoltotals labelfield=change_name label=ALL
```

See also

[addtotals](#), [stats](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `addcoltotals` command.

addinfo

Synopsis

Add fields that contain common information about the current search.

Syntax

```
| addinfo
```

Description

Adds global information about the search to each event. Currently the following fields are added:

- `info_min_time`: the earliest time bound for the search
- `info_max_time`: the latest time bound for the search
- `info_sid`: ID of the search that generated the event
- `info_search_time`: time when the search was executed.

Examples

Example 1: Add information about the search to each event.

```
... |addinfo
```

See also

[search](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `adinfo` command.

addtotals

Synopsis

Computes the sum of all numeric fields for each result.

Syntax

```
addtotals [row=bool] [col=bool] [labelfield=field] [label=string] [fieldname=field]  
field-list
```

Required arguments

field-list

Syntax: <field>...

Description: One or more numeric fields, delimited with a space, and can include wildcards.

Optional arguments

row

Datatype: <bool>

Description: Specifies whether to compute the arithmetic sum of *field-list* for each result. Defaults to true.

col

Datatype: <bool>

Description: Specifies whether to add a new result (a summary event) that represents the sum of each field. Defaults to false.

fieldname

Datatype: <field>

Description: If *row*=true, use this to specify the name of the field to put the sum.

label

Datatype: <string>

Description: If `labelfield` is specified, it will be added to this summary event with the value set by the 'label' option.

`labelfield`

Datatype: <field>

Description: If `col=true`, use this to specify a name for the summary event.

Description

The default `addtotals` command (`row=true`) computes the arithmetic sum of all numeric fields that match *field-list* (wildcarded field list). If list is empty all fields are considered. The sum is placed in the specified field or `total` if none was specified.

If `col=t`, `addtotals` 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 `| addtotals col=true`, you can use the [addcoltotals](#) command to calculate a summary event.

Examples

Example 1: Compute the sums of the numeric fields of each results.

```
... | addtotals
```

Example 2: Calculate the sums of the numeric fields of each result, and put the sums in the field "sum".

```
... | addtotals fieldname=sum
```

Example 3: Compute the sums of the numeric fields that match the given list, and save the sums in the field "sum".

```
... | addtotals fieldname=sum foobar* *baz*
```

Example 4: Compute the sums of all the fields, and put the sums in a summary event called "change_name".

```
... | addtotals col=t labelfield=change_name label=ALL
```

See also

[stats](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `addtotals` command.

analyzefields

Synopsis

Analyzes numerical fields for their ability to predict another discrete field.

Syntax

```
af | analyzefields classfield=field
```

Required arguments

`classfield`

Syntax: `classfield=<field>`

Description: For best results, `classfield` should have 2 distinct values, although multi-class analysis is possible.

Description

Using *field* as a discrete random variable, analyze all *numerical* fields to determine the ability for each of those fields to `predict` the value of the `classfield`. For best results, `classfield` should have 2 distinct values, although multi-class analysis is possible.

The `analyzefields` command returns a table with five columns: `field`, `count`, `cocur`, `acc`, and `balacc`.

- `field` is the name of the field in the search results.
- `count` is the number of occurrences of the field in the search results.
- `cocur` is the `cocurrence` of the field versus the `classfield`. The `cocur` is 1 if `field` exists in every event that has `classfield`.

- `acc` is the accuracy in predicting the value of the `classfield` using the value of the field. This is only valid for numerical fields.
- `balacc`, or "balanced accuracy", is the non-weighted average of the accuracies in predicted each value of the `classfield`. This is only valid for numerical fields.

Examples

Example 1: Analyze the numerical fields to predict the value of "is_activated".

```
... | af 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

Use the `anomalies` command to look for events that you don't expect to find based on the values of a field in a sliding set of events. The `anomalies` command assigns an unexpectedness score to each event in a new field named `unexpectedness`. Whether the event is considered anomalous or not depends on a `threshold` value that is compared against the calculated unexpectedness score. The event is considered unexpected or anomalous if the unexpectedness `> threshold`.

Note: After you run `anomalies` in the timeline Search view, add the `unexpectedness` field to your events list using the **Pick fields** menu.

Synopsis

Computes an `unexpectedness` score for an event.

Syntax

```
anomalies [threshold=num] [labelonly=bool] [normalize=bool] [maxvalues=int]
[field=field] [blacklist=filename] [blacklistthreshold=num] [by-clause]
```


Optional arguments

threshold

Datatype: threshold=<num>

Description: A number to represent the unexpectedness limit. If an event's calculated unexpectedness is greater than this limit, the event is considered unexpected or anomalous. Defaults to 0.01.

labelonly

Datatype: labelonly=<bool>

Description: Specify how you want to output to be returned. The `unexpectedness` field is appended to all events. If set to true, no events are removed. If set to false, events that have a `unexpected` score less than the threshold (boring events) are removed. Defaults to false.

normalize

Datatype: normalize=<bool>

Description: Specify whether or not to normalize numeric values. For cases where `field` contains numeric data that should not be normalized, but treated as categories, set `normalize=false`. Defaults to true.

maxvalues

Datatype: maxvalues=<int>

Description: Specify the size of the sliding window of previous events to include when determining the unexpectedness of an event's field value. This number is between 10 and 10000. Defaults to 100.

field

Datatype: field=<field>

Description: The field to analyze when determining the unexpectedness of an event. Defaults to `_raw`.

blacklist

Datatype: blacklist=<filename>

Description: A name of a CSV file of events that is located in `$SPLUNK_HOME/var/run/splunk/BLACKLIST.csv`. Any incoming event that is similar to an event in the blacklist is treated as not anomalous (that is, uninteresting) and given an unexpectedness score of 0.0.

blacklistthreshold

Datatype: blacklistthreshold=<num>

Description: Specify similarity score threshold for matching incoming events to blacklisted events. If the incoming event has a similarity score

above the `blacklistthreshold`, it is marked as unexpected. Defaults to 0.05.

by clause

Syntax: by <fieldlist>

Description: Used to specify a list of fields to segregate results for anomaly detection. For each combination of values for the specified field(s), events with those values are treated entirely separately.

Description

For those interested in how the unexpected score of an event is calculated, the algorithm is proprietary, but roughly speaking, it is based on the similarity of that event (X) to a set of previous events (P):

$$\text{unexpectedness} = [s(P \text{ and } X) - s(P)] / [s(P) + s(X)]$$

Here, `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 and also normalizes for the differing event sizes.

You can run the `anomalies` command again on the results of a previous `anomalies`, to further narrow down the results. As each run operates over 100 events, the second call to `anomalies` is approximately running over a window of 10,000 previous events.

Examples

Example 1: This example just shows how you can tune the search for anomalies using the `threshold` value.

```
index=_internal | anomalies by group | search group=*
```

This search looks at events in the `_internal` index and calculates the `unexpectedness` score for sets of events that have the same `group` value. This means that the sliding set of events used to calculate the `unexpectedness` for each unique `group` value will only include events that have the same `group` value. The `search` command is then used to show only events that include the `group` field. Here's a snapshot of the results:

8	6/9/11 6:45:25.889 AM	06-09-2011 06:45:25.889 -0700 INFO Metrics - group=pipeline, name=fschangemanager, processor=sendindex, cpu_seconds=0.000000, executes=1, cumulative_hits=54 unexpectedness=0.015679 group=pipeline
9	6/9/11 6:45:25.889 AM	06-09-2011 06:45:25.889 -0700 INFO Metrics - group=pipeline, name=fschangemanager, processor=fschangemanager, cpu_seconds=0.000000, executes=1, cumulative_hits=54 unexpectedness=0.036496 group=pipeline
10	6/9/11 6:40:46.845 AM	06-09-2011 06:40:46.845 -0700 INFO Metrics - group=pipeline, name=dev-null, processor=nullqueue, cpu_seconds=0.000000, executes=1, cumulative_hits=3105 unexpectedness=0.031136 group=pipeline
11	6/9/11 6:33:01.849 AM	06-09-2011 06:33:01.849 -0700 INFO Metrics - group=per_source_thruput, series="/applications/splunk/var/log/splunk/splunkd.log", kbps=0.014680, eps=0.129032, kb=0.455078, ev=4, avg_age=1.000000, max_age=1 unexpectedness=0.010084 group=per_source_thruput

With the default `threshold=0.01`, you can see that some of these events may be very similar. This next search increases the `threshold` a little:

```
index=_internal | anomalies threshold=0.03 by group | search group=*
```





1	6/9/11 7:15:49.396 AM	06-09-2011 07:15:49.396 -0700 INFO Metrics - group=pipeline, name=fschangemanager, processor=fschangemanager, cpu_seconds=0.000000, executes=1, cumulative_hits=57 unexpectedness=0.033217 group=pipeline
2	6/9/11 7:05:29.712 AM	06-09-2011 07:05:29.712 -0700 INFO Metrics - group=pipeline, name=fschangemanager, processor=fschangemanager, cpu_seconds=0.000000, executes=1, cumulative_hits=56 unexpectedness=0.033159 group=pipeline
3	6/9/11 6:55:12.120 AM	06-09-2011 06:55:12.120 -0700 INFO Metrics - group=pipeline, name=fschangemanager, processor=fschangemanager, cpu_seconds=0.000000, executes=1, cumulative_hits=55 unexpectedness=0.034783 group=pipeline
4	6/9/11 6:54:41.488 AM	06-09-2011 06:54:41.488 -0700 INFO Metrics - group=realtime_search_data, sid=rt_1307627653.17, mean_preview_period=1.022148 unexpectedness=0.118321 group=realtime_search_data

With the higher `threshold` value, you can see at-a-glance that there is more distinction between each of the events (the timestamps and key/value pairs).

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 value. This has to be done explicitly because the `threshold` attribute of the `anomalies` command is not a field. The `eval` command is then used to define another new field, `score`, that is either "anomalous" or "boring" based on how the `unexpectedness` compares to the `threshold` value. Here's a snapshot of these results:

5		6/9/11 8:00:38.712 AM	06-09-2011 08:00:38.712 -0700 INFO Metrics - group=search_concurrency, system total, active_hist_searches=20, active_realtime_searches=0 unexpectedness=0.947826 ▾ group=search_concurrency ▾ score=anomalous ▾
6		6/9/11 8:00:38.712 AM	06-09-2011 08:00:38.712 -0700 INFO Metrics - group=realtime_search_data, system total, drop_count=0 unexpectedness=0.920792 ▾ group=realtime_search_data ▾ score=anomalous ▾
7		6/9/11 8:00:38.712 AM	06-09-2011 08:00:38.712 -0700 INFO Metrics - group=queue, name=typingqueue, max_size_kb=500, current_size_kb=0, current_size=0, largest_size=31, smallest_size=0 unexpectedness=0.025237 ▾ group=queue ▾ score=boring ▾
8		6/9/11 8:00:38.712 AM	06-09-2011 08:00:38.712 -0700 INFO Metrics - group=queue, name=tcpin_queue, max_size_kb=500, current_size_kb=0, current_size=0, largest_size=0, smallest_size=0 unexpectedness=0.029412 ▾ group=queue ▾ score=boring ▾

More examples

Example 1: Show most interesting events first, ignoring any in the blacklist 'boringevents'.

```
... | anomalies blacklist=boringevents | sort -unexpectedness
```

Example 2: Use with transactions to find regions of time that look unusual.

```
... | transaction maxpause=2s | anomalies
```

Example 3: Look for anomalies in each source separately -- a pattern in one source will not affect that it is anomalous in another source.

```
... | anomalies by source
```

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.

anomalousvalue

Synopsis

Finds and summarizes irregular, or uncommon, search results.

Syntax

```
anomalousvalue <av-option> [action] [pthresh] [field-list]
```

Required arguments

<av-option>

Syntax: minsupcount=<integer> | maxanofreq=<float> |
minsupfreq=<float> | minnormfreq=<float>

Description: Fields that occur only in a couple of events aren't very informative (which one of three values is anomalous?). minsupcount, maxanofreq, minsupfreq, and minnormfreq set thresholds to filter out these uninformative fields.

- `maxanofreq=p` Omits a field from consideration if more than a fraction `p` of the events that it appears in would be considered anomalous.
- `minnormfreq=p` Omits a field from consideration if less than a fraction `p` of the events that it appears in would be considered normal.
- `minsupcount=N` Specifies that a field must appear in at least `N` of the events `anomalousvalue` processes to be considered for deciding which fields are anomalous.
- `minsupfreq=p` Identical to `minsupcount`, but instead of specifying an absolute number `N` of events, specify a minimum fraction of events `p` (between 0 and 1).

Optional arguments

action

Syntax: action=annotate | filter | summary

Description: Specify whether to return the anomaly score (annotate), filter out events with anomalous values (filter), or a summary of anomaly statistics (summary). Defaults to filter.

- If action is `annotate`, a new field is added to the event containing the anomalous value that indicates the anomaly score of the value.
- If action is `filter`, events with anomalous value(s) are isolated.
- If action is `summary`, a table summarizing the anomaly statistics for each field is generated.

field-list

Syntax: <field>, ...

Description: List of fields to consider.

pthresh

Syntax: pthresh=<num>

Description: Probability threshold (as a decimal) that has to be met for a value to be considered anomalous. Defaults to 0.01.

Description

The `anomalousvalue` command looks at the entire event set and considers the distribution of values when deciding if a value is anomalous or not. 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.

Examples

Example 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
.
```

Example 2: Return uncommon values from the host "reports".

```
host="reports" | anomalousvalue action=filter pthresh=0.02
```

Example 3: Return a summary of the anomaly statistics for each numeric field.

```
source=/var/log* | anomalousvalue action=summary pthresh=0.02 | search
isNum=YES
```

11 results in the last 24 hours (from 1:00:00 PM May 4 to 1:16:46 PM May 5, 2011)											
« prev 1 2 next » Options...											
Results per page 10 ▾											
Overlay: None ▾											
	catAnoFreq% ▾	catNormFreq% ▾	count ▾	distinct_count ▾	fieldname ▾	isNum ▾	mean ▾	numAnoFreq% ▾	stdev ▾	supportFreq% ▾	useCat ▾ useNum ▾
1	0.0000	4.1667	1360	24	date_hour	YES	11.704412	0.0000	6.485571	100.0000	YES YES
2	0.0000	50.0000	1360	2	date_mday	YES	4.552941	0.0000	0.497189	100.0000	YES YES
3	0.0000	1.6667	1360	60	date_minute	YES	29.972059	0.0000	17.499810	100.0000	YES YES
4	0.0000	1.6667	1359	60	date_second	YES	29.378219	0.0000	17.190094	99.9265	YES YES
5	0.0000	100.0000	1360	1	date_year	YES	2011.000000	0.0000	0.000000	100.0000	NO NO
6	0.0000	100.0000	30	2	err	YES	0.000000	0.0000	0.000000	2.2059	NO NO
7	0.1471	99.8529	1360	3	linecount	YES	1.019118	0.1471	0.628667	100.0000	YES YES
8	0.0000	100.0000	557	1	pid	YES	0.000000	0.0000	0.000000	40.9559	NO NO
9	0.0000	100.0000	29	1	securitySessionID	YES	0.000000	0.0000	0.000000	2.1324	NO NO
10	0.1471	33.2843	1360	5	timeendpos	YES	15.467647	6.8382	0.721812	100.0000	YES NO

See also

[af](#), [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.

append

Use the `append` command to append the results of a subsearch to the results of your current search. The `append` command will run only over historical data; it will not produce correct results if used in a real-time search.

Synopsis

Appends **subsearch** results to current results.

Syntax

```
append [subsearch-options]* subsearch
```

Required arguments

subsearch

Description: A search pipeline. Read more about how subsearches work in the *Search manual*.

Optional arguments

subsearch-options

Syntax: maxtime=<int> | maxout=<int> | timeout=<int>

Description: Controls how the subsearch is executed.

Subsearch options

maxtime

Syntax: maxtime=<int>

Description: The maximum time (in seconds) to spend on the subsearch before automatically finalizing. Defaults to 60.

maxout

Syntax: maxout=<int>

Description: The maximum number of result rows to output from the subsearch. Defaults to 50000.

timeout

Syntax: timeout=<int>

Description: The maximum time (in seconds) to wait for subsearch to fully finish. Defaults to 120.

Description

Append the results of a subsearch to the current search as new results at the end of current results.

Examples

Example 1

This example uses recent (October 18-25, 2010) earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains the source network (Src), ID (Eqid), version, date, location, magnitude, depth (km) and number of reporting stations (NST) for each earthquake over the last 7 days.

Download the text file, **M 2.5+ earthquakes, past 7 days**, save it as a CSV file, and upload it to Splunk. Splunk should extract the fields automatically. Note that you'll be seeing data from the 7 days previous to your download, so your results will vary from the ones displayed below.

Count the number of earthquakes that occurred in and around California yesterday and then calculate the total number of quakes.

```
source="eqs7day-M1.csv" Region="*California" | stats count by Region |  
append [search source="eqs7day-M1.csv" Region="*California" | stats  
count]
```

This example searches for all the earthquakes in the California regions (Region="*California"), then counts the number of earthquakes that occurred in each separate region.

The `stats` command doesn't let you count the total number of events at the same time as you count the number of events split-by a field, so 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 is listed as the last row in the column:

6 results yesterday (during Sunday, October 24, 2010)

Options... Results per page 10

Overlay: None

	Region	count
1	Central California	23
2	Greater Los Angeles area, California	1
3	Northern California	18
4	San Francisco Bay area, California	1
5	Southern California	13
6		56

This search basically demonstrates using the `append` command similar to the `addcoltotals` command, to add the column totals.

Example 2

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Count the number of different customers who purchased something from the Flower & Gift shop yesterday, and break this count down by the type of product (Candy, Flowers, Gifts, Plants, and Balloons) 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 category_id
| append [search sourcetype=access_* action=purchase | top 1 clientip
by category_id] | table category_id, 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 (`category_id`).

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 (`category_id`), 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`).

Overlay:

	category_id ↕	dc(clientip) ↕	clientip ↕	count ↕
1	BALLOONS	217		
2	CANDY	206		
3	FLOWERS	298		
4	GIFTS	293		
5	PLANTS	196		
6	BALLOONS		192.0.1.55	13
7	CANDY		189.222.1.45	10
8	FLOWERS		10.32.1.39	15
9	GIFTS		189.222.1.47	13
10	PLANTS		187.231.45.35	18

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 doesn't let you manipulate or reformat the output.

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 **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Count the number of different IP addresses who 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`).

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:

Overlay:

	method ↕	dc(clientip) ↕	count ↕	clientip ↕	percent ↕
1	GET	313	8778		
2	POST	241	499		
3	GET		55	187.231.45.53	0.626566
4	POST		5	233.77.49.32	1.002004

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.

More examples

Example 1: Append the current results with the tabular results of "fubar".

```
... | chart count by bar | append [search fubar | chart count by baz]
```

See also

[appendcols](#), [join](#), [set](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the append command.

appendcols

Synopsis

Appends the fields of the **subsearch** results to current results, first results to first result, second to second, etc.

Syntax

```
appendcols [override=bool|subsearch-options]* subsearch
```

Required arguments

subsearch

Description: A search pipeline. Read more about how subsearches work in the *Search manual*.

Optional arguments

override

Datatype: <bool>

Description: If option override is false (default), if a field is present in both a subsearch result and the main result, the main result is used.

subsearch-options

Syntax: maxtime=<int> | maxout=<int> | timeout=<int>

Description: Controls how the subsearch is executed.

Subsearch options

maxtime

Syntax: maxtime=<int>

Description: The maximum time (in seconds) to spend on the subsearch before automatically finalizing. Defaults to 60.

maxout

Syntax: maxout=<int>

Description: The maximum number of result rows to output from the subsearch. Defaults to 50000.

timeout

Syntax: timeout=<int>

Description: The maximum time (in seconds) to wait for subsearch to fully finish. Defaults to 120.

Description

Appends fields of the results of the subsearch into input search results by combining the external fields of the subsearch (fields that do not start with '__') into the current results. The first subsearch result is merged with the first main result, the second with the second, and so on. If option `override` is false (default), if a field is present in both a subsearch result and the main result, the main result is used. If it is true, the subsearch result's value for that field is used.

Examples

Example 1: Search for "404" events and append the fields in each event to the previous search results.

```
... | appendcols [search 404]
```

See also

[append](#), [join](#), [set](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `appendcols` command.

appendpipe

Synopsis

Appends the result of the subpipeline applied to the current result set to results.

Syntax

appendpipe [run_in_preview=<bool>] [<subpipeline>]

Arguments

run_in_preview

Syntax: run_in_preview=T|F

Description: Specify whether or not to run the command in preview mode. Defaults to T.

Examples

Example 1: Append subtotals for each action across all users.

```
index=_audit | stats count by action user | appendpipe [stats sum(count)
as count by action | eval user = "ALL USERS"] | sort action
```

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.

associate

The `associate` 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, it has an entropy of zero. If it has multiple values, the more evenly those values are distributed, the higher the entropy.

Synopsis

Identifies correlations between fields.

Syntax

associate [*associate-option*]* [*field-list*]

Optional arguments

associate-option

Syntax: supcnt | supfreq | improv

Description: Options for the associate command.

field-list

Syntax: <field>, ...

Description: List of fields, non-wildcarded. If a list of fields is provided, analysis will be restricted to only those fields. By default all fields are used.

Associate options

supcnt

Syntax: supcnt=<num>

Description: Specify the minimum number of times that the "reference key=reference value" combination must appear. Must be a non-negative integer. Defaults to 100.

supfreq

Syntax: supfreq=<num>

Description: Specify the minimum frequency of "reference key=reference value" combination as a fraction of the number of total events. Defaults to 0.1.

improv

Syntax: improv=<num>

Description: Specify a limit, or minimum entropy improvement, for the "target key". The resulting calculated entropy improvement, which is the difference between the unconditional entropy (the entropy of the target key) and the conditional entropy (the entropy of the target key, when the reference key is the reference value) must be greater than or equal to this

limit. Defaults to 0.5.

Description

The `associate` command outputs a table with columns that include the fields that are analyzed (`Reference_Key`, `Reference_Value`, and `Target_Key`), the entropy that is calculated for each pair of field values (`Unconditional_Entropy`, `Conditional_Entropy`, and `Entropy_Improvement`), and a message that summarizes the relationship between the fields values that is deduced based on the entropy calculation (`Description`).

The `Description` is intended as a user-friendly representation of the result, and is written in the format: "When the '`Reference_Key`' has the value '`Reference_Value`', the entropy of '`Target_Key`' decreases from `Unconditional_Entropy` to `Conditional_Entropy`."

Examples

Example 1: This example demonstrates how you might analyze the relationship of fields in your web access logs.

```
sourcetype=access_* NOT status=200 | fields method, status | associate  
| 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 a lot of fields and you can use the `associate` command to see a relationship between all pairs of fields and values in your data. To simplify this example, we restrict the search to two fields: `method` and `status`. Also, the `associate` command outputs a number of columns (see `Description`) that, for now, we won't go into; so, we use the `table` command to display only the columns we want to see. The result looks something like this:

5 results over all time

20 per page

Overlay: None

	Reference_Key	Reference_Value	Target_Key	Top_Conditional_Value	Description
1	method	POST	status	302 (74.58% -> 100.00%)	When 'method' has the value 'POST', the entropy of 'status' decreases from 1.224 to 0.000.
2	status	301	method	GET (25.42% -> 100.00%)	When 'status' has the value '301', the entropy of 'method' decreases from 0.818 to 0.000.
3	status	302	method	POST (74.58% -> 100.00%)	When 'status' has the value '302', the entropy of 'method' decreases from 0.818 to 0.000.
4	status	304	method	GET (25.42% -> 100.00%)	When 'status' has the value '304', the entropy of 'method' decreases from 0.818 to 0.000.
5	status	404	method	GET (25.42% -> 100.00%)	When 'status' has the value '404', the entropy of 'method' decreases from 0.818 to 0.000.

For this particular result set, (you can see in the Fields area, to the left of the results area) there are:

- two `method` values: POST and GET
- five `status` values: 301, 302, 304, 404, and 503

The first row of the results tells you that when `method=POST`, the `status` field is 302 for all of those events. The `associate` command concludes that, if `method=POST`, the `status` is likely to be 302. You can see this same conclusion in the third row, which references `status=302` to predict the value of `method`.

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, and 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 "CV (FRV% -> FCV%)" where CV is the conditional Value, FRV 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.

Note: This example uses sample data from the Splunk Tutorial. which you can download and add to run this search and see these results. For more information, refer to "Get the sample data into Splunk" in the Tutorial.

Example 2: Return results associated with each other (that have at least 3 references to each other).

```
index=_internal sourcetype=splunkd | associate supcnt=3
```

Example 3: 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

[correlate](#), [contingency](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `associate` command.

audit

Synopsis

Returns audit trail information that is stored in the local audit index.

Syntax

audit

Description

View audit trail information stored in the local `audit` index. Also decrypt signed audit events while checking for gaps and tampering.

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

Synopsis

Sets up data for calculating the moving average.

Syntax

autoregress *field* [AS <newfield>] [p=<p_start>[-<p_end>]]

Required arguments

field-list

Syntax: <field>...

Description: One or more numeric fields, delimited with a space, and can not include wildcards.

Optional arguments

p

Syntax: p=<int:p_start>

Description: If 'p' option is unspecified, it is equivalent to p_start = p_end = 1 (i.e., copy only the previous one value of *field* into *field_p1*)

newfield

Syntax: <field>

Description: note that p cannot be a range if newfield is specified.

p_start

Syntax: <int>

Description: If 'p' option is unspecified, it is equivalent to p_start = p_end = 1 (i.e., copy only the previous one value of *field* into *field_p1*)

p_end

Syntax: <int>

Description: If 'p' option is unspecified, it is equivalent to p_start = p_end = 1 (i.e., copy only the previous one value of *field* into *field_p1*)

Description

Sets up data for auto-regression (moving average) by copying the p-th previous values for *field* into each event as *newfield* (or if unspecified, new fields **field_pp-val** for *p-val* = *p_start-p_end*). If 'p' option is unspecified, it is equivalent to p_start = p_end = 1 (i.e., copy only the previous one value of *field* into *field_p1*). Note that p cannot be a range if newfield is specified.

Examples

Example 1: For each event, copy the 3rd previous value of the 'foo' field into the field 'oldfoo'.

```
... | autoregress foo AS oldfoo p=3
```

Example 2: For each event, copy the 2nd, 3rd, 4th, and 5th previous values of the 'count' field into the respective fields 'count_p2', 'count_p3', 'count_p4', and 'count_p5'.

```
... | autoregress count p=2-5
```

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.

bucket

Synopsis

Puts continuous numerical values into discrete sets.

Syntax

```
bucket [<bucketing-option>]* <field> [as <field>]
```

Required arguments

<field>

Datatype: <field>

Description: Specify a field name.

Optional arguments

<bucketing-option>

Datatype: bins | minspan | span | start-end

Description: Discretization options. See "Bucketing options" for details.

<newfield>

Datatype: <string>

Description: A new name for the field.

Bucketing 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 bucket, 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 buckets. Data outside of the [start, end] range is discarded.

Log span syntax

<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).

Span length syntax

span-length

Syntax: [<timescale>]

Description: A span length based on time.

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.

<sec>

Syntax: s | sec | secs | second | seconds

Description: Time scale in seconds.

<min>

Syntax: m | min | mins | minute | minutes

Description: Time scale in minutes.

<hr>

Syntax: h | hr | hrs | hour | hours

Description: Time scale in hours.

<day>

Syntax: d | day | days

Description: Time scale in days.

<month>

Syntax: mon | month | months

Description: Time scale in months.

<subseconds>

Syntax: us | ms | cs | ds

Description: Time scale in microseconds (us), milliseconds (ms), centiseconds (cs), or deciseconds (ds).

Description

Puts continuous numerical values in fields into discrete sets, or buckets. The default field processed is `_time`. Note: Bucket is called by chart and timechart automatically and is only needed for statistical operations that timechart and chart cannot process.

Examples

Example 1: Return the average "thruput" of each "host" for each 5 minute time span.

```
... | bucket _time span=5m | stats avg(thruput) by _time host
```

Example 2: Bucket search results into 10 bins, and return the count of raw events for each bucket.

```
... | bucket size bins=10 | stats count(_raw) by size
```

See also

[chart](#), [timechart](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the bucket command.

bucketdir

Synopsis

Replaces a field value with higher-level grouping, such as replacing filenames with directories.

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: Specify either "/" or "\" as the separating character. This depends on the operating system.

Description

Returns at most 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 SEP character. The default pathfield=source; sizefield=totalCount; maxcount=20; countfield=totalCount; sep="/" or "\", depending on the os.

Examples

Example 1: Get 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 bucket command.

chart

Use the `chart` command to create charts that can display any series of data that you want to plot. You can decide what field is tracked on the x-axis of the chart. The `chart`, `timechart`, `stats`, `eventstats`, and `streamstats` are all designed to work in conjunction with statistical functions. To find more information about statistical functions and how they're used, see "[Functions for stats, chart, and timechart](#)" in the Search Reference Manual.

Synopsis

Returns results in a tabular output for charting.

Syntax

```
chart [sep=<string>] [cont=<bool>] [limit=<int>] [agg=<stats-agg-term>] (  
<stats-agg-term> | <sparkline-agg-term> | <eval-expression>...) [ by <field>
```

(<bucketing-option>)... [<split-by-clause>] | [over <field>
(<bucketing-option>)... (by <split-by-clause>]]

For a list of chart functions with descriptions and examples, see ["Functions for stats, chart, and timechart"](#).

Required arguments

agg

Syntax: agg=<stats-agg-term>

Description: Specify an aggregator or function. For a list of stats functions with descriptions and examples, see ["Functions for stats, chart, and timechart"](#).

sparkline-agg-term

Syntax: <sparkline-agg> [AS <wc-field>]

Description: A sparkline specifier optional renamed to a new field.

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 [Functions for eval](#). 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 may not produce valid results if the values are not numerical. Additionally, Splunk can concatenate the two operands if they are both strings. When concatenating values with '.', Splunk treats both values as strings regardless of their actual type.

Optional arguments

agg

Syntax: <stats-agg-term>

Description: For a list of stats functions with descriptions and examples, see ["Functions for stats, chart, and timechart"](#).

bucketing-option

Syntax: bins | span | <start-end>

Description: Discretization options. If a bucketing option is not supplied, `timechart` defaults to `bins=300`. This finds the smallest bucket size that results in no more than 300 distinct buckets. For more bucketing options, see the [bucket command reference](#).

cont

Syntax: <bool>

Description: Specifies whether its continuous or not.

limit

Syntax: <int>

Description: Specify a limit for series filtering; limit=0 means no filtering.

single-agg

Syntax: count|<stats-func>(<field>)

Description: A single aggregation applied to a single field (can be eval'd field). No wildcards are allowed. The field must be specified, except when using the special `count` aggregator that applies to events as a whole.

sep

Syntax: sep=<string>

Description: Used to construct output field names when multiple data series are used in conjunctions with a split-by field.

split-by-clause

Syntax: <field> (<tc-option>)* [<where-clause>]

Description: Specifies a field to split by. If field is numerical, default discretization is applied; discretization is defined with `tc-option`.

Stats functions

stats-agg-term

Syntax: <stats-func>(<eval'd-field> | <wc-field>) [AS <wc-field>]

Description: A statistical specifier optionally renamed to a new field name. The specifier can be by an aggregation function applied to a field or set of fields or an aggregation function applied to an arbitrary eval expression.

stats-function

Syntax: avg() | c() | count() | dc() | distinct_count() | earliest() | estdc() | estdc_error() | exactperc<int>() | first() | last() | latest() | list() | max() | median() | min() | mode() | p<int>() | perc<int>() | range() | stdev() | stdevp() | sum() | sumsq() | upperperc<int>() | values() | var() | varp()

Description: Functions used with the stats command. Each time you invoke the `stats` command, you can use more than one function; however, you can only use one `by` clause. For a list of stats functions with descriptions and examples, see ["Functions for stats, chart, and timechart"](#).

Sparkline function 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. 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.

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 bucket.

Bucketing options

bins

Syntax: bins=<int>

Description: Sets the maximum number of bins to discretize into.

span

Syntax: span=<log-span> | span=<span-length>

Description: Sets the size of each bucket, 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 buckets. Data outside of the [start, end] range is discarded.

Log span syntax

<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).

Span length syntax

span-length

Syntax: [<timescale>]

Description: A span length based on time.

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.

<sec>

Syntax: s | sec | secs | second | seconds

Description: Time scale in seconds.

<min>

Syntax: m | min | mins | minute | minutes

Description: Time scale in minutes.

<hr>

Syntax: h | hr | hrs | hour | hours

Description: Time scale in hours.

<day>

Syntax: d | day | days

Description: Time scale in days.

<month>

Syntax: mon | month | months

Description: Time scale in months.

<subseconds>

Syntax: us | ms | cs | ds

Description: Time scale in microseconds (us), milliseconds (ms), centiseconds (cs), or deciseconds (ds).

tc options

tc-option

Syntax: <bucketing-option> | usenull=<bool> | useother=<bool> | nullstr=<string> | otherstr=<string>

Description: Options for controlling the behavior of splitting by a field.

usenull

Syntax: usenull=<bool>

Description: Controls whether or not a series is created for events that do not contain the split-by field.

nullstr

Syntax: nullstr=<string>

Description: If usenull is true, this series is labeled by the value of the nullstr option, and defaults to NULL.

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>.

otherstr

String: otherstr=<string>

Description: If useother is true, this series is labeled by the value of the otherstr option, and defaults to OTHER.

where clause

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.

<where-comp>

Syntax: <wherein-comp> | <wherethresh-comp>

Description: A criteria for the where clause.

<wherein-comp>

Syntax: (in|notin) (top|bottom)<int>

Description: A where-clause criteria that requires the aggregated series value be in or not in some top or bottom grouping.

<wherethresh-comp>

Syntax: (<|>)()?<num>

Description: A where-clause criteria that requires the aggregated series value be greater than or less than some numeric threshold.

Description

Create tabular data output suitable for charting. The x-axis variable is specified with a *by field* and is discretized if necessary. Charted fields are converted to numerical quantities if necessary.

Whereas `timechart` generates a chart with `_time` as the x-axis, `chart` produces a table with an arbitrary field as the x-axis. In addition, `chart` allows for a split-by field. When such a field is included, the output will be a table where each column represents a distinct value of the split-by field.

This is in contrast with `stats`, where each row represents a single unique combination of values of the group-by fields. The number of columns to be included is by default limited to 10, but can be adjusted by the inclusion of an optional where clause. See *where-clause* for a more detailed description.

Chart allows for an eval-expression, which is required to be renamed unless a split-by clause is present. You can also specify the 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=0` means no series filtering. The `limit` and `agg` options are ignored if an explicit where clause is provided.

A note about split-by fields

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:

```
... | chart sum(A) by A span=log2
```

However, you can work around this with an `eval` expression, for example:

```
... | eval A1=A | chart sum(A) by A1 span=log2
```

Examples

Example 1

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 **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Chart the number of different page requests, GET and POST, that occurred for each Web server.

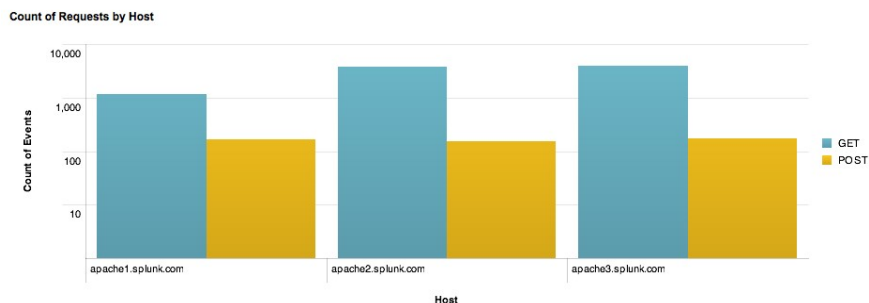
```
sourcetype=access_* | chart count(eval(method="GET")) AS GET,
count(eval(method="POST")) AS POST by host
```

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, it renames the field that represents these results to "GET" (this is what the "AS" is doing). The second clause does the same for POST events. The counts of both types of events are then separated by the Web server, indicated by the `host` field, from which they appeared.

This returns the following table:

	host ↕	GET ↕	POST ↕
1	apache1.splunk.com	1152	169
2	apache2.splunk.com	3771	154
3	apache3.splunk.com	3855	176

Click **Show report** to format the chart in Report Builder. Here, the y-axis is shown on a logarithmic scale:



This chart displays the total count of events for each event type, GET or POST, based on the `host` value. The logarithmic scale is used for the y-axis because of the difference in range of values between the number of GET and POST events.

Note: You can use the `stats`, `chart`, and `timechart` commands to perform the same statistical calculations on your data. The `stats` command returns a table of results. The `chart` command returns the same table of results, but you can use the Report Builder to format this table as a chart. If you want to chart your results over a time range, use the `timechart` command. You can also see variations of this example with the [chart](#) and [timechart](#) commands.

Example 2

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **All time**.

Create a chart to show the number of transactions based on their duration (in seconds).

```
sourcetype=access_* 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`). This produces the following table:

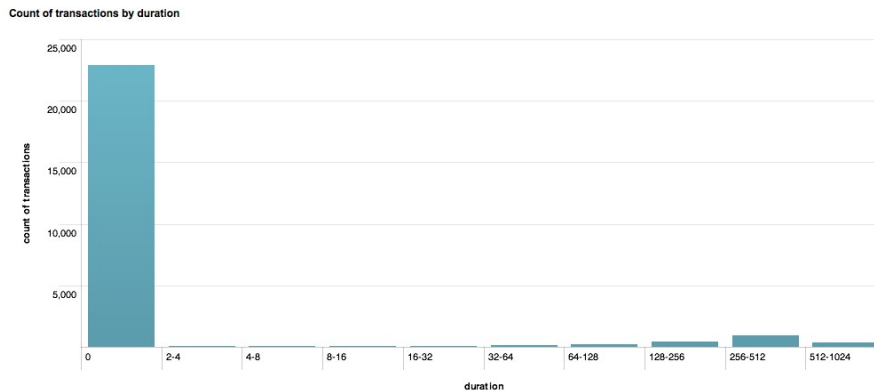
10 results over all time

Options... Results per page 20

Overlay: None

	duration	count
1	0	22900
2	2-4	3
3	4-8	6
4	8-16	21
5	16-32	56
6	32-64	130
7	64-128	240
8	128-256	461
9	256-512	957
10	512-1024	359

Click **Show report** to format the chart in Report Builder. Here, it's formatted as a column chart:



As you would expect, most transactions take between 0 and 2 seconds to complete. Here, it looks like the next greater number of transactions spanned between 256 and 512 seconds (approximately, 4-8 minutes). (In this case however, the numbers may be a bit extreme because of the way that the data was generated.)

Example 3

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **All time**.

Create a chart to show the average number of events in a transaction based on the duration of the transaction.


```
sourcetype=access_* action=purchase | transaction clientip maxspan=10m
| chart avg(eventcount) by duration span=log2
```

This example uses the same transaction defined in Example 2. The `transaction` command also creates a new field called `eventcount`, which is the number of events in a single transaction.

The transactions are then piped into the `chart` command and 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 of \log_2 (`span=log2`). This produces the following table:

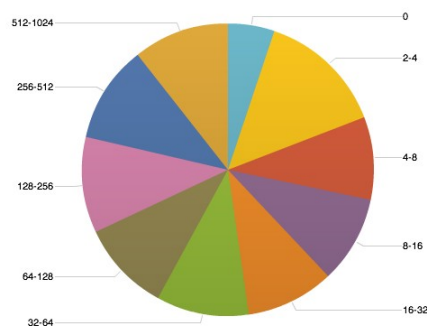
10 results over all time

Options... Results per page 20

Overlay: None

	duration ↕	avg(eventcount) ↕
1	0	1.598210
2	2-4	4.333333
3	4-8	2.833333
4	8-16	3.000000
5	16-32	3.017857
6	32-64	3.184615
7	64-128	3.116667
8	128-256	3.279826
9	256-512	3.335423
10	512-1024	3.292479

Click **Show report** to format the chart in Report Builder. Here, it's formatted as a pie chart:



Each wedge of the pie chart represents the average number of events in the transactions of the corresponding duration. After you create the pie chart, you can mouseover each of the sections to see these values (in Splunk Web).

Example 4

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Chart how many different people bought something and what they bought at the Flower & Gift shop Yesterday.

```
sourcetype=access_* action=purchase | chart dc(clientip) over date_hour  
by category_id 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 `category_id` 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.

This produces the following table:

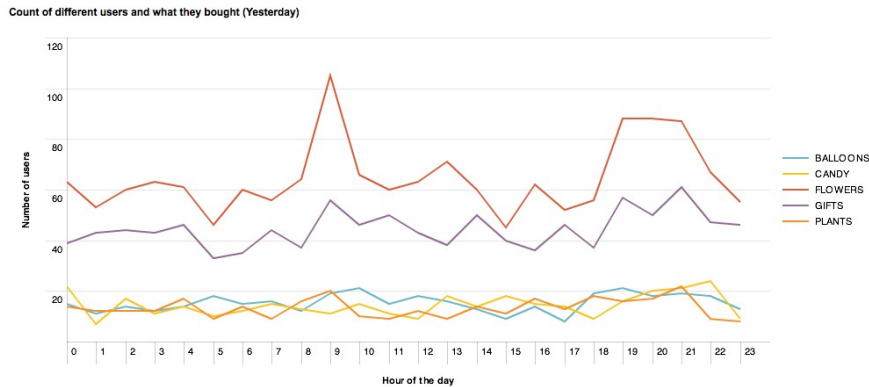
24 results yesterday (during Sunday, October 3, 2010)

« prev 1 2 next » | Options... Results per page 20 ▾

Overlay: None ▾

	date_hour ▾	BALLOONS ▾	CANDY ▾	FLOWERS ▾	GIFTS ▾	PLANTS ▾
1	0	15	22	63	39	14
2	1	11	7	53	43	12
3	2	14	17	60	44	12
4	3	12	11	63	43	12
5	4	14	14	61	46	17
6	5	18	10	46	33	9
7	6	15	12	60	35	14
8	7	16	15	56	44	9
9	8	12	13	64	37	16
10	9	19	11	105	56	20

Click **Show report** to format the chart in Report Builder. Here, it's formatted as a line chart:



Each line represents a different type of product that is sold at the Flower & Gift shop. 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 flowers. Most of the purchases were made early in the day, around lunch time, and early in the evening.

Example 5

This example uses recent (September 29-October 6, 2010) earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains the source network (Src), ID (Eqid), version, date, location, magnitude, depth (km) and number of reporting stations (NST) for each earthquake over the last 7 days.

Download the text file, **M 1+ earthquakes, past 7 days**, and upload it to Splunk. Splunk should extract the fields automatically.

Create a chart that shows the number of earthquakes and the magnitude of each one that occurred in and around California.

```
source=eqs7day-M1.csv Region=*California | chart count over Magnitude
by Region useother=f
```

This search counts the number of earthquakes that occurred in the the California regions. The count is then broken down for each region based on the magnitude of the quake. Because the `Region` value is non-numeric, the search uses the `useother=f` argument to exclude events that don't match.

This produces the following table:

22 results over all time

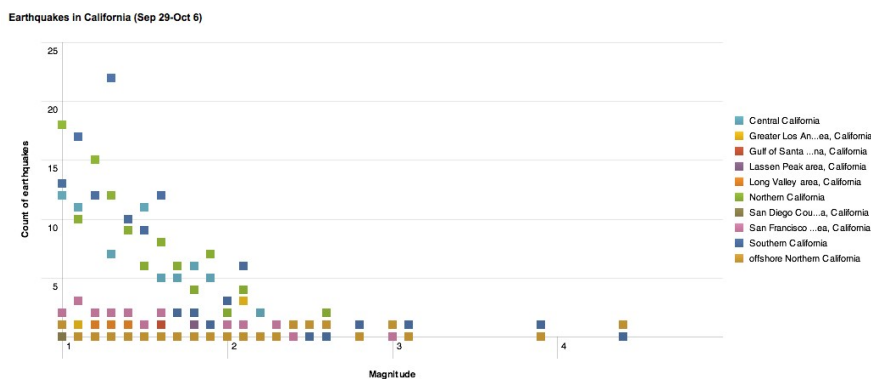
« prev 1 2 next » | Options...

Results per page 20

Overlay: None

	Magnitude	Central California	Greater Los Angeles area, California	Gulf of Santa Catalina, California	Lassen Peak area, California	Long Valley area, California
1	1.0	12	0	0	0	0
2	1.1	11	1	0	0	0
3	1.2	12	1	0	0	1
4	1.3	7	1	0	0	1
5	1.4	9	0	0	0	1
6	1.5	11	0	0	0	0
7	1.6	5	1	1	0	0

Click **Show report** to format the chart in Report Builder. Here, it's formatted as a scatter chart:



This chart shows that the majority of the quakes that occurred in the past week were of magnitudes between 1 and 2.2. Quakes of higher magnitude were less frequent--Yay!

Also, the plot points for each region may overlap with another region's plot. If you want to see just the points for one region at a time, mouseover the region in the legend. If you want to see metrics for an individual point, mouseover that point on the chart. A tooltip will open and display the corresponding `Magnitude`, `Region`, and `count` of earthquakes.

More examples

Example 1: Return `max(delay)` for each value of `foo`.

```
... | chart max(delay) over foo
```

Example 2: Return `max(delay)` for each value of `foo` split by the value of `bar`.

```
... | chart max(delay) over foo by bar
```

Example 3: Return 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
```

Example 4: Return the the maximum "delay" by "size", where "size" is broken down into a maximum of 10 equal sized buckets.

```
... | chart max(delay) by size bins=10
```

Example 5: Return the average (mean) "size" for each distinct "host".

```
... | chart avg(size) by host
```

See also

[timechart](#), [bucket](#), [sichart](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the chart command.

cluster

You can use the `cluster` command to learn more about your data and to find common and/or rare events in your data. For example, if you are investigating an IT problem and you don't know specifically what to look for, use the `cluster` command to find anomalies. In this case, anomalous events are those that aren't 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.

Synopsis

Cluster similar events together.

Syntax

```
cluster [slc-option]*
```

Optional arguments

slc-option

Syntax: t=<num> | delims=<string> | showcount=<bool> | countfield=<field> | labelfield=<field> | field=<field> | labelonly=<bool> | match=(termlist | termset | ngramset)

Description: Options for configuring simple log clusters (slc).

SLC options

t

Syntax: 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 is 0.8.

delims

Syntax: 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: showcount=<bool>

Description: Shows the size of each cluster. Default is true, unless `labelonly` is set to true. When `showcount=false`, each indexer clusters its own events before clustering on the search head.

countfield

Syntax: countfield=<field>

Description: Name of the field to write the cluster size to. The cluster size is the count of events in the cluster. Defaults to `cluster_count`.

labelfield

Syntax: labelfield=<field>

Description: Name of the field to write the cluster number to. Splunk counts each cluster and labels each with a number as it groups events into clusters. Defaults to `cluster_label`.

field

Syntax: field=<field>

Description: Name of the field to analyze in each event. Defaults to `__raw`.

labelonly

Description: labelonly=<bool>

Syntax: Specifies whether reduce each cluster to a single representative cluster. If true, keeps all original events and labels them with a cluster number (the value of `labelfield`). If false, reduces each cluster to a single event. to keep all original events instead of reducing each cluster to a single event. Defaults to `false`.

match

Syntax: match=(termlist | termset | ngramset)

Description: Specify 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`. Defaults to `termlist`.

Description

The `cluster` command groups events together based on how similar they are to each other. Unless you specify a different field, `cluster` uses the `_raw` field to break down the events into terms (`match=termlist`) 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.

Examples

Example 1

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 2

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 3

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](#), [cluster](#), [kmeans](#), [outlier](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `cluster` command.

collect

Synopsis

Puts search results into a summary index.

Syntax

collect *index* [*arg-options*]*

Required arguments

index

Syntax: index=<string>

Description: Name of the index where Splunk should add the events. The index must exist for events to be added to it, 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>

Description: Optional arguments for the collect command.

Collect options

addtime

Syntax: addtime=<bool>

Description: If the search results you want to collect do not have a `_raw` field (such as results of [stats](#), [chart](#), [timechart](#)), specify whether to prefix a time field into each event. Specifying false means that Splunk will use its generic date detection against fields in whatever order they happen to be in the summary rows. Specifying true means that Splunk will use the search time range `info_min_time` (which is added by [sistats](#)) or `_time`. Splunk adds the time field based on the first field that it finds: `info_min_time`, `_time`, `now()`. Default is true.

file

Syntax: file=<string>

Description: Name of the file where to write the events. Optional, default "<random-num>_events.stash". The following placeholders can be used in

the file name `$timestamp$`, `$random$` and will be replaced with a timestamp and a random number, respectively.

".stash" needs to be added at the end of the file name when used with "index=", if not the data will be added to the main index.

marker

Syntax: `marker=<string>`

Description: A string, usually of key-value pairs, to append to each event written out. Optional, default is empty.

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. Defaults to false.

spool

Syntax: `spool=<bool>`

Description: If set to true (default is true), the summary indexing file will be written to Splunk's spool directory, where it will be indexed automatically. If set to false, file will be written to `$SPLUNK_HOME/var/run/splunk`.

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. (defaults to false)

Description

Adds the results of the search into the specified index. Behind the scenes, the events are written to a file whose name format is:

`events_<random-num>.stash`, unless overwritten, in a directory which is watched for new events by splunk. If the events contain a `_raw` field, then the raw field is saved; if the events don't have a `_raw` field, one is constructed by concatenating all the fields into a comma separated `key=value` pairs list.

Note: The `collect` command also works with all-time real-time searches.

Examples

Example 1: Put "download" events into an index named "downloadcount".

```
eventtype tag="download" | collect index=downloadcount
```

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

Synopsis

Given a duration field, finds the number of "concurrent" events for each event.

Syntax

```
concurrency duration=<field> [start=<field>] [output=<field>]
```

Required arguments

duration

Syntax: duration=<field>

Description: A field that represents a span of time.

Optional arguments

start

Syntax: start=<field>

Description: A field that represents the start time. Default is `_time`.

output

Syntax: output=<field>

Description: A field to write the resulting number of concurrent events. Default is "concurrency".

Description

Concurrency is the number of events that occurred simultaneously at the *start time* of the event, not the number of events that occurred during any overlap.

An event X is concurrent with event Y if $(X.start, X.start + X.duration)$ overlaps at all with: $(Y.start, Y.start + Y.duration)$

Examples

Example 1

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **All time**.

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="*signon*" endswith="purchase" | concurrency
duration=duration | eval duration=toString(duration,"duration")
```

This example groups events into transactions if they have the same values of `JSESSIONID` and `clientip`, defines an event as the beginning of the transaction if it contains the string "signon" and the last event of the transaction if it contains the string "purchase".

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.

Transaction ID	Start Time	End Time	Duration	Concurrency
1	10/18/10 4:18:00.000 AM	10/18/10 5:25:17 AM	01:07:17	1
2	10/18/10 4:52:18.000 AM	10/18/10 5:21:42 AM	00:29:24	2

These results show that the first transaction started at 4:18 AM, lasted 1 hour 7 minutes and 17 seconds, and has a `concurrency` of 1. The concurrency number is inclusive, so this means that this was the only transaction taking place at 4:18 AM.

The second transaction started at 4:52:18 AM. At this time, the first transaction was still taking place, so the concurrency for this transaction is 2.

Example 2

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Use the time between each purchase to count the number of different purchases that occurred at the same time.

```
sourcetype=access_* action=purchase | delta _time AS timeDelta p=1 |
eval timeDelta=abs(timeDelta) | concurrency duration=timeDelta
```

This example 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)`). This `timeDelta` is then used as the `duration` for calculating concurrent events.

1	10/24/10 12:00:00.000 AM	10.32.1.36 - - [24/Oct/2010:00:00:00] "GET /flower_store/category.screen?category_id=CANDY HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-21&JSESSIONID=SD8SL9FF7ADFF8" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3056 1178 JSESSIONID=SD8SL9FF7ADFF8 clientip=10.32.1.36 timeDelta=49 concurrency=1
2	10/24/10 12:00:00.000 AM	10.32.1.36 - - [24/Oct/2010:00:00:00] "GET /flower_store/category.screen?category_id=CANDY HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-21&JSESSIONID=SD8SL9FF7ADFF8" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3571 2615 JSESSIONID=SD8SL9FF7ADFF8 clientip=10.32.1.36 timeDelta=0 concurrency=2
3	10/24/10 12:00:49.000 AM	178.19.3.31 - - [24/Oct/2010:00:00:49] "GET /flower_store/category.screen?category_id=FLOWERS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-6&JSESSIONID=SD8SL9FF7ADFF8" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 1351 499 JSESSIONID=SD8SL9FF7ADFF8 clientip=178.19.3.31 timeDelta=11 concurrency=1

These results show that the first and second purchases occurred at the same time. However, the first purchase has a `concurrency=1`. The second purchase has a `concurrency=2`, which includes itself and the first purchase event. Notice that the third purchase has a `concurrency=1`. This is because by the time of that

purchase, 12:49 AM, the first purchase (which had `timeDelta=49` seconds) already completed.

Example 3

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Use the time between each consecutive transaction to calculate the number of transactions that occurred at the same time.

```
sourcetype=access_* | transaction JSESSIONID clientip
startswith="*signon*" endswith="purchase" | delta _time AS timeDelta
p=1 | eval timeDelta=abs(timeDelta) | concurrency duration=timeDelta |
eval timeDelta=toString(timeDelta,"duration")
```

This example groups events into transactions if they have the same values of `JSESSIONID` and `clientip`, defines an event as the beginning of the transaction if it contains the string "signon" and the last event of the transaction if it contains the string "purchase".

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.

```
2 10/24/10 12:23:25.000 AM 177.23.21.54 - - [24/Oct/2010:00:23:25] "POST /flower_store/j_signon_check HTTP/1.1" 302 309
"http://mystore.splunk.com/flower_store/enter_order_information.screen&JSESSIONID=SD8SL9FF7ADFF8" "Mozilla/5.0
(X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 2609 2018
177.23.21.54 - - [24/Oct/2010:00:23:25] "GET /flower_store/category.screen?category_id=FLOWERS HTTP/1.1" 200
10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-1&JSESSIONID=SD8SL9FF7ADFF8"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos
Firefox/1.5.0.10" 3813 2252
JSESSIONID=SD8SL9FF7ADFF8 | clientip=177.23.21.54 | duration=00:00:00 | timeDelta=00:00:35 | concurrency=1

3 10/24/10 12:24:00.000 AM 187.231.45.47 - - [24/Oct/2010:00:24:00] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" 200
10567 "http://mystore.splunk.com/flower_store/signon_welcome.screen&JSESSIONID=SD8SL9FF7ADFF8" "Mozilla/5.0 (X11;
U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3295 1582
187.231.45.47 - - [24/Oct/2010:00:55:36] "GET /flower_store/product.screen?product_id=FL-DLH-02 HTTP/1.1" 200
10989 "http://mystore.splunk.com/flower_store/category.screen?category_id=FLOWERS&JSESSIONID=SD8SL9FF7ADFF8"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos
Firefox/1.5.0.10" 1260 31
187.231.45.47 - - [24/Oct/2010:00:55:36] "GET /flower_store/category.screen?category_id=FLOWERS HTTP/1.1" 200
10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-1&JSESSIONID=SD8SL9FF7ADFF8"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos
Firefox/1.5.0.10" 1244 604
JSESSIONID=SD8SL9FF7ADFF8 | clientip=187.231.45.47 | duration=00:31:36 | timeDelta=00:02:15 | concurrency=1
```

Unlike Example 1, which was run over **All time**, this search was run over the time range **Other > Yesterday**. There were no concurrent transactions for these first two transactions.

Example 4

This example uses recent (October 18-25, 2010) earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains the source network (Src), ID (Eqid), version, date, location, magnitude, depth (km) and number of reporting stations (NST) for each earthquake over the last 7 days.

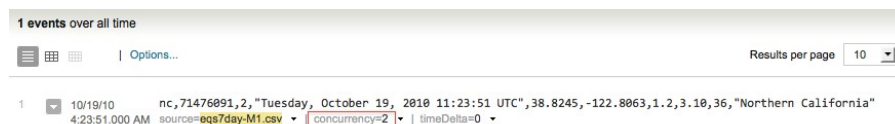
Download the text file, **M 2.5+ earthquakes, past 7 days**, save it as a CSV file, and upload it to Splunk. Splunk should extract the fields automatically. Note that you'll be seeing data from the 7 days previous to your download, so your results will vary from the ones displayed below.

Search for recent earthquakes in and around California that occurred at the same time.

```
source="eqs7day-M1.csv" Region="*California" | delta _time AS timeDelta  
p=1 | eval timeDelta=abs(timeDelta) | concurrency duration=timeDelta |  
where concurrency>1
```

This example starts off with a search for all the earthquakes in the California area (`Region="*California"`). Then it calculates the time between each earthquake and the one before using the `delta` command. The absolute value of this change in time, `timeDelta`, is then used as the `duration` value to find concurrent earthquakes.

The events are piped into the `where` command to filter out events that don't have a concurrent event (`concurrency>1`).



1 events over all time	
1	10/19/10 4:23:51.000 AM nc,71476091,2,"Tuesday, October 19, 2010 11:23:51 UTC",38.8245,-122.8063,1.2,3.10,36,"Northern California"



Here, this result shows you that there were two earthquakes (`concurrency=2`) that occurred on Tuesday, October 19, 2010 at 11:23:51 UTC (this is the `Dateime` value). But, this only shows 1 event.

What if you want to see more information about these events? For example, what was the magnitude of these two quakes and where did they occur? You have the `Datetime` value and can search for that over the time range, **All time**. Or you can use a subsearch:

```
source="eqs7day-M1.csv" [search source="eqs7day-M1.csv"  
Region="*California" | delta _time AS timeDelta p=1 | eval  
timeDelta=abs(timeDelta) | concurrency duration=timeDelta | where
```

```
concurrency>1 | table Datetime]
```

The original search is run first as a subsearch that uses the `table` command to return only the `Datetime` of the results. This `Datetime` is used in the outer search and returns the following events:

2 events over all time	
  Options...	Results per page 10
1	<div><div>10/19/10 4:23:51.000 AM</div><div>nc,71476886,4,"Tuesday, October 19, 2010 11:23:51 UTC",38.8245,-122.8055,1.1,3.36,61,"Northern California" Region=Northern California Lat=38.8245 Lon=-122.8055 Magnitude=1.1</div></div>
2	<div><div>10/19/10 4:23:51.000 AM</div><div>nc,71476891,2,"Tuesday, October 19, 2010 11:23:51 UTC",38.8245,-122.8063,1.2,3.18,36,"Northern California" Region=Northern California Lat=38.8245 Lon=-122.8063 Magnitude=1.2</div></div>

Now, you can see that the two concurrent events occurred in Northern California, fairly close together, and with magnitudes of 1.1 and 1.2 respectively.

More examples

Example 1: Calculate the number of concurrent events for each event and emit as field 'foo':

```
... | concurrency duration=total_time output=foo
```

Example 2: Calculate the number of concurrent events using the 'et' field as the start time and 'length' as the duration:

```
... | concurrency duration=length start=et
```

See also

[timechart](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `concurrency` command.

contingency

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.

Synopsis

Builds a contingency table for two fields.

Syntax

`contingency [<contingency-option>]* <field> <field>`

Required arguments

`<field>`

Syntax: `<field>`

Description: Any field, non wildcarded.

Optional arguments

`contingency-option`

Syntax: `<maxopts> | <mincover> | <usetotal> | <totalstr>`

Description: Options for the contingency table.

Contingency option

`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 will be ignored. A value of 0 means unlimited rows or columns. By default, `maxrows=0` and `maxcols=0`.

mincover

Syntax: mincolcover=<num> | minrowcover=<num>

Description: Specify the minimum percentage of values for the row or column field. If the number of entries needed to cover the required percentage of values exceeds `maxrows` or `maxcols`, `maxrows` or `maxcols` takes precedence. By default, `mincolcover=1.0` and `minrowcover=1.0`.

usetotal

Syntax: usetotal=<bool>

Description: Specify whether or not to add row and column totals. Default is `usetotal=true`.

totalstr

Syntax: totalstr=<field>

Description: Field name for the totals row and column. Default is `totalstr=TOTAL`.

Description

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` parameters. 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.

Examples

Example 1

Build a contingency table to see if there is a relationship between the values of `log_level` and `component`.

```
index=_internal | contingency log_level component maxcols=5
```

	log_level ↕	Metrics ↕	TailingProcessor ↕	FileClassifierManager ↕	TcpinputFd ↕	UserManagerPro ↕	TOTAL ↕
1	INFO	632830	11860	0	0	0	646098
2	WARN	0	5	11845	0	0	11988
3	ERROR	0	0	0	772	406	1184
4	WARNING	0	0	0	0	0	26
5	DEBUG	0	0	0	0	0	8
6	TOTAL	632830	11865	11845	772	406	659305

These results show you at-a-glance what components, if any, may be causing issues in your Splunk instance. The `component` field has many values (>50), so

this example, uses `maxcols` to show only five of the values.

Example 2

Build a contingency table to see the installer download patterns from users based on the platform they are running.

```
host="download"| contingency name platform
```

	name ↕	Windows ↕	Linux ↕	MacOS ↕	Solaris ↕	FreeBSD ↕	AIX ↕	HPUX ↕	TOTAL ↕
1	windows_installer	640	0	0	0	0	0	0	640
2	linux_installer	0	378	0	0	0	0	0	378
3	osx_installer	0	0	114	0	0	0	0	114
4	solaris_installer	0	0	0	50	0	0	0	50
5	freebsd_installer	0	0	0	0	12	0	0	12
6	aix_installer	0	0	0	0	0	3	0	3
7	hpux_installer	0	0	0	0	0	0	2	2
8	TOTAL	640	378	114	50	12	3	2	1199

This is pretty straightforward because you don't expect users running one platform to download an installer file for another platform. Here, the `contingency` command just confirms that these particular fields are not independent. If this chart showed otherwise, for example if a great number of Windows users downloaded the OSX installer, you might want to take a look at your web site to make sure the download resource is correct.

Example 3

This example uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains the source network (Src), ID (Eqid), version, date, location, magnitude, depth (km) and number of reporting stations (NST) for each earthquake over the last 7 days.

Download the text file, **M 2.5+ earthquakes, past 7 days**, save it as a CSV file, and upload it to Splunk. Splunk should extract the fields automatically. Note that you'll be seeing data from the 7 days previous to your download, so your results will vary from the ones displayed below. (Here, the CSV file is uploaded to the custom index `recentquakes`. Also, the file includes two weeks of data.)

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).

Build a contingency table to look at the relationship between the magnitudes and depths of recent earthquakes.

```
index=recentquakes | contingency Magnitude Depth | sort Magnitude
```

This search is very simple. But because there are quite a range of values for the `Magnitude` and `Depth` fields, the results is a very large matrix. Before building the table, we want to reformat the values of the field:

```
index=recentquakes | eval Magnitude=case(Magnitude<=1, "0.0 - 1.0",
Magnitude>1 AND Magnitude<=2, "1.1 - 2.0", Magnitude>2 AND Magnitude<=3,
"2.1 - 3.0", Magnitude>3 AND Magnitude<=4, "3.1 - 4.0", Magnitude>4 AND
Magnitude<=5, "4.1 - 5.0", Magnitude>5 AND Magnitude<=6, "5.1 - 6.0",
Magnitude>6 AND Magnitude<=7, "6.1 - 7.0", Magnitude>7, "7.0+") | eval
Depth=case(Depth<=70, "Shallow", Depth>70 AND Depth<=300, "Mid",
Depth>300 AND Depth<=700, "Deep") | contingency Magnitude Depth | sort
Magnitude
```

Now, the search uses the `eval` command with the `case()` function to redefine the values of `Magnitude` and `Depth`, bucketing them into a range of values. For example, the `Depth` values are redefined as "Shallow", "Mid", or "Deep". This creates a more readable table:

	Magnitude ↕	Shallow ↕	Mid ↕	Deep ↕	TOTAL ↕
1	0.0 - 1.0	64	2	0	66
2	1.1 - 2.0	509	37	0	546
3	2.1 - 3.0	257	33	0	290
4	3.1 - 4.0	95	14	0	109
5	4.1 - 5.0	145	42	10	197
6	5.1 - 6.0	48	5	4	57
7	6.1 - 7.0	3	0	0	3
8	TOTAL	1121	133	14	1268

There were a lot of quakes in this 2 week period. 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 magnitude ranges were shallow. And, there are significantly fewer earthquakes in the mid-to-high range. In this data set, the deep-focused quakes were all in the mid-range of magnitudes.

See also

[associate](#), [correlate](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `contingency` command.

convert

The convert command converts field values into numerical values. Alternatively, you can use [functions of the eval command](#) such as `strftime()`, `strptime()`, or `tostring()`.

Synopsis

Converts field values into numerical values.

Syntax

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

Required arguments

<convert-function>

Syntax: auto() | ctime() | dur2sec() | memk() | mktime() | mstime() | none()
| num() | rmcomma() | rmunit()

Description: Functions for convert.

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, refer to the topic "[Common time format variables](#)". Defaults to `%m/%d/%y %H:%M:%S`.

<new_fieldname>

Syntax: <string>

Description: Rename function to a new field.

Convert functions

auto()

Syntax: auto(<wc-field>)

Description: Automatically convert the field(s) 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.

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.

dur2sec()

Syntax: dur2sec(<wc-field>)

Description: Convert a duration format "D+HH:MM:SS" to seconds.

memk()

Syntax: memk(<wc-field>)

Description: Convert a {KB, MB, GB} denominated size quantity into a KB.

mktime()

Syntax: mktime(<wc-field>)

Description: Convert an human readable time string to an epoch time. Use `timeformat` option to specify exact format to convert from.

mstime()

Syntax: mstime(<wc-field>)

Description: Convert a MM:SS.SSS format to seconds.

none()

Syntax: none(<wc-field>)

Description: In the presence of other wildcards, indicates that the matching fields should not be converted.

num()

Syntax: num(<wc-field>)

Description: Like `auto()`, except non-convertible values are removed.

rmcomma()

Syntax: rmcomma(<wc-field>)

Description: Removes all commas from value, for example `rmcomma(1,000,000.00)` returns `1000000.00`.

rmunit()

Syntax: rmunit(<wc-field>)

Description: Looks for numbers at the beginning of the value and removes trailing text.

Description

Converts the values of fields into numerical values. When renaming a field using `AS`, the original field is left intact.

Examples

Example 1

This example uses sendmail email server logs and refers to the logs with `sourcetype=sendmail`. The sendmail logs have two duration fields, `delay` and `xdelay`.

The `delay` is the total amount of time a message took to deliver or bounce. The `delay` is expressed as "D+HH:MM:SS", which indicates the time it took in hours (HH), minutes (MM), and seconds (SS) to handle delivery or rejection of the message. If the delay exceeds 24 hours, the time expression is prefixed with the number of days and a plus character (D+).

The `xdelay` is the total amount of time the message took to be transmitted during final delivery, and its time is expressed as "HH:MM:SS".

Change the sendmail duration format of `delay` and `xdelay` to seconds.

```
sourcetype=sendmail | convert dur2sec(delay) dur2sec(xdelay)
```

This search pipes all the sendmail events into the `convert` command and uses the `dur2sec()` function to convert the duration times of the fields, `delay` and `xdelay`, into seconds.

Here is how your search results will look after you use the **fields sidebar** to add the fields to your events:

1	4/8/10	Apr 8 11:01:15 splunk3 sendmail[31035]: n38111xA031035: to=root, ctladdr=root (0/0), delay=00:00:14, xdelay=00:00:00, mailer=relay, pri=30443, relay=[127.0.0.1] [127.0.0.1], dsn=2.0.0, stat=Sent (n3811FXt031097 Message accepted for delivery)
		sourcetype=sendmail delay=14 xdelay=0
2	4/8/10	Apr 8 10:01:11 splunk3 sendmail[16285]: n38H11pk016285: to=root, ctladdr=root (0/0), delay=00:00:10, xdelay=00:00:00, mailer=relay, pri=30443, relay=[127.0.0.1] [127.0.0.1], dsn=2.0.0, stat=Sent (n38H18t1016334 Message accepted for delivery)
		sourcetype=sendmail delay=10 xdelay=0
3	4/8/10	Apr 8 09:27:10 splunk3 sendmail[7922]: n38GQt21007855: to=<spam@splunkit.com>, ctladdr=<spam@splunkit.com> (501/502), delay=00:00:11, xdelay=00:00:03, mailer=local, pri=30863, dsn=2.0.0, stat=Sent
		sourcetype=sendmail delay=11 xdelay=3
4	4/8/10	Apr 8 09:01:10 splunk3 sendmail[1335]: n38G11UH001335: to=root, ctladdr=root (0/0), delay=00:00:09, xdelay=00:00:01, mailer=relay, pri=30443, relay=[127.0.0.1] [127.0.0.1], dsn=2.0.0, stat=Sent (n38G19pD001366 Message accepted for delivery)
		sourcetype=sendmail delay=9 xdelay=1

You can compare the converted field values to the original field values in the events list.

Example 2

This example uses syslog data.

Convert a UNIX epoch time to a more readable time formatted to show hours, minutes, and seconds.

```
sourcetype=syslog | convert timeformat="%H:%M:%S" ctime(_time) AS  
c_time | table _time, c_time
```

The `ctime()` function converts the `_time` value of `syslog` (`sourcetype=syslog`) events to the format specified by the `timeformat` argument. The `timeformat="%H:%M:%S"` arguments tells Splunk to format the `_time` value as HH:MM:SS.

Here, the `table` command is used to show the original `_time` value and the converted time, which is renamed `c_time`:

Overlay:

	<code>_time</code> ↕	<code>c_time</code> ↕
1	10/19/10 11:58:26.000 PM	23:58:26
2	10/19/10 11:56:10.000 PM	23:56:10
3	10/19/10 11:53:54.000 PM	23:53:54
4	10/19/10 11:48:15.000 PM	23:48:15
5	10/19/10 11:45:59.000 PM	23:45:59
6	10/19/10 11:45:59.000 PM	23:45:59
7	10/19/10 11:43:43.000 PM	23:43:43
8	10/19/10 11:41:27.000 PM	23:41:27
9	10/19/10 11:39:12.000 PM	23:39:12
10	10/19/10 11:36:56.000 PM	23:36:56

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.

Example 3

This example uses syslog data.

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` value of `syslog` (`sourcetype=syslog`) events from a minutes and seconds to just seconds.

Here, the `table` command is used to show the original `_time` value and the converted time, which is renamed `ms_time`:

Overlay:

	<u>time</u> ⚡	ms_time ⚡
1	10/19/10 11:58:26.000 PM	1287557906
2	10/19/10 11:56:10.000 PM	1287557770
3	10/19/10 11:53:54.000 PM	1287557634
4	10/19/10 11:48:15.000 PM	1287557295
5	10/19/10 11:45:59.000 PM	1287557159
6	10/19/10 11:45:59.000 PM	1287557159
7	10/19/10 11:43:43.000 PM	1287557023
8	10/19/10 11:41:27.000 PM	1287556887
9	10/19/10 11:39:12.000 PM	1287556752
10	10/19/10 11:36:56.000 PM	1287556616

The `mstime()` function changes the timestamp to a numerical value. This is useful if you want to use it for more calculations.

More examples

Example 1: Convert values of the "duration" field into number value by removing string values in the field value. For example, if "duration="212 sec"", the resulting value will be "duration="212"".

```
... | convert rmunit(duration)
```

Example 2: Change the sendmail syslog duration format (D+HH:MM:SS) to seconds. For example, if "delay="00:10:15"", the resulting value will be "delay="615"".

```
... | convert dur2sec(delay)
```

Example 3: Change all memory values in the "virt" field to Kilobytes.

```
... | convert memk(virt)
```

Example 4: Convert every field value to a number value except for values in the field "foo" (use the "none" argument to specify fields to ignore).

```
... | convert auto(*) none(foo)
```

Example 5: Example usage

```
... | convert dur2sec(xdelay) dur2sec(delay)
```

Example 6: Example usage

```
... | convert auto(*)
```

See also

[eval](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `convert` command.

correlate

This page is currently a work in progress; expect frequent near-term updates.

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 that represents the percentage of times that the two fields exist in the same events.

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.

Synopsis

Calculates the correlation between different fields.

Syntax

```
correlate [type=cocur] [_metainclude=<bool>]
```

Optional arguments

`type`

Syntax: `type=cocur`

Description: Type of correlation to calculate. Currently the only available options is the co-occurrence matrix, which contains the percentage of times that two fields exist in the same events. Cell values of 1.0 indicate that the two fields always exist together in the data.

`_metainclude`

Syntax: `_metainclude=<bool>`

Description: This is an internal option. Specifies whether to include the internal metadata fields (that start with '_') in the analysis. Defaults to `false`.

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:

RowField ↕	abandoned_channels ↕	actions_triggered ↕	active_hist_searches ↕	active_realtime_searches ↕	average_kbps ↕	avg_age ↕
1 abandoned_channels	1.00	0.00	0.00	0.00	0.00	0.00
2 actions_triggered	0.00	1.00	0.00	0.00	0.00	0.00
3 active_hist_searches	0.00	0.00	1.00	1.00	0.00	0.00
4 active_realtime_searches	0.00	0.00	1.00	1.00	0.00	0.00
5 average_kbps	0.00	0.00	0.00	0.00	1.00	0.00
6 avg_age	0.00	0.00	0.00	0.00	0.00	1.00
7 browser	0.00	0.00	0.00	0.00	0.00	0.00
8 bytes	0.00	0.00	0.00	0.00	0.00	0.00

Because there are difference types of logs in the `_internal`, you can expect to see that many 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`, etc. 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 all the field value pairs will have a correlation of 1.0 (co-occur in 100% of the search results). For these download events, you might be able to spot an issue depending on which pair 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.

crawl

Synopsis

Crawls the filesystem for files of interest to Splunk.

Syntax

```
crawl [ files | network ] [crawl-option]*
```

Optional arguments

crawl-option

Syntax: <string>=<string>

Description: Override settings from crawl.conf.

Description

Crawls for the discovery of new sources to index. Default crawl settings are found in crawl.conf and crawl operations are logged to \$splunk_home/var/log/splunk/crawl.log. Generally to be used in conjunction with the `input` command. Specify crawl options to override settings in crawl.conf. Note: If you add crawl to a search, Splunk only returns data it generates from crawl. Splunk doesn't return any data generated before `| crawl`.

Examples

Example 1: Crawl root and home directories and add all possible inputs found (adds configuration information to "inputs.conf").

```
| crawl root="/;/Users/" | input add
```

Example 2: Crawl bob's home directory.

```
| crawl root=/home/bob
```

Example 3: Add all sources found in bob's home directory to the 'preview' index.

```
| crawl root=/home/bob | input add index=preview
```

Example 4: Crawl using default settings defined in crawl.conf.

```
| crawl
```

See also

[input](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the crawl command.

dbinspect

Synopsis

Returns information about the Splunk index.

Syntax

```
dbinspect [index=<string>] [<span>|<timeformat>]
```

Optional arguments

index

Syntax: index=<string>

Description: Specify the name of the index to inspect.

Syntax: span=<int>|<int><timescale>

Description: Specify the span length of the bucket. If using a timescale unit (sec, min, hr, day, month, or subseconds), this is used as a time range. If not, this is an absolute bucket "length".

<timeformat>

Syntax: timeformat=<string>

Description: Set the time format. Defaults to
`timeformat=%m/%d/%Y:%H:%M:%S.`

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.

`<sec>`

Syntax: `s` | `sec` | `secs` | `second` | `seconds`

Description: Time scale in seconds.

`<min>`

Syntax: `m` | `min` | `mins` | `minute` | `minutes`

Description: Time scale in minutes.

`<hr>`

Syntax: `h` | `hr` | `hrs` | `hour` | `hours`

Description: Time scale in hours.

`<day>`

Syntax: `d` | `day` | `days`

Description: Time scale in days.

`<month>`

Syntax: `mon` | `month` | `months`

Description: Time scale in months.

`<subseconds>`

Syntax: `us` | `ms` | `cs` | `ds`

Description: Time scale in microseconds (`us`), milliseconds (`ms`), centiseconds (`cs`), or deciseconds (`ds`).

Description

The `dbinspect` command returns information about the Splunk index that you specify.

When you invoke the `dbinspect` command without a bucket span, Splunk returns the following information about the given index's buckets: `earliestTime`,

eventCount, hostCount, id, latestTime, modTime, path, rawSizeMB, sizeOnDiskMB, sourceCount, sourceTypeCount, and state.

When you invoke the `dbinspect` command with a bucket span, Splunk returns a chartable representation of the spans of each bucket.

Examples

Example 1: Display a chart with the span size of 1 day.

```
| dbinspect index=_internal span=1d
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `dbinspect` command.

dedup

Synopsis

Removes the subsequent duplicate results that match specified criteria.

Syntax

```
dedup [<N>] <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.

Optional arguments

consecutive

Syntax: consecutive=<bool>

Description: Specify whether to only remove duplicate events that are consecutive (true). Defaults to false.

keepempty

Syntax: keepempty=<bool>

Description: If an event contains a null value for one or more of the specified fields, the event is either retained (true) or discarded. Defaults to false.

keepevents

Syntax: keepevents=<bool>

Description: When true, keeps all events and removes specific values. Defaults to false.

<N>

Syntax: <int>

Description: Specify the first N (where N > 0) number of events to keep, for each combination of values for the specified field(s). The non-option parameter, if it is a number, is interpreted as N.

<sort-by-clause>

Syntax: (- | +) <sort-field>

Description: List of fields to sort by and their order, descending (-) or ascending (+).

Sort field options

<sort-field>

Syntax: <field> | auto(<field>) | str(<field>) | ip(<field>) | num(<field>)

Description: Options for sort-field.

<field>

Syntax: <string>

Description: The name of the field to sort.

auto

Syntax: auto(<field>)

Description: Determine automatically how to sort the field's values.

ip

Syntax: ip(<field>)

Description: Interpret the field's values as an IP address.

num

Syntax: num(<field>)

Description: Treat the field's values as numbers.

str

Syntax: str(<field>)

Description: Order the field's values lexicographically.

Description

The `dedup` command lets you specify the number of duplicate events to keep based on the values of a field. The event returned for the `dedup` field will be the first event found (most recent in time). If you specify a number, `dedup` interprets this number as the count of duplicate events to keep, `N`. If you don't specify a number, `N` is assumed to be 1 and it keeps only the first occurring event and removes all consecutive duplicates.

The `dedup` command also lets you sort by some list of fields. This will remove all the duplicates and then sort the results based on the specified sort-by field. Note, that this will only be valid or effective if your search returns multiple results. The other options let you specify other criteria, for example you may want to keep all events, but for events with duplicate values, remove those values instead of the entire event.

Note: We do not recommend that you run the `dedup` command against the `_raw` field if you are searching over a large volume of data. Doing this causes Splunk to add a map of each unique `_raw` value seen which will impact your search performance. This is expected behavior.

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.

```
... | dedup source sortby +_time
```

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 -_size
```

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

Synopsis

Performs a deletion from the index.

Syntax

delete

Description

Piping a search to the delete operator marks all the events returned by that search so that future searches do not return them. No user (even with admin permissions) will be able to see this data using Splunk. **Currently, piping to delete does not reclaim disk space.**

Note: Splunk does not let you run the `delete` operator during a real-time search; you cannot delete events as they come in. If you try to use `delete` during a real-time search, Splunk will display an error.

The delete operator can only be accessed by a user with the "delete_by_keyword" **capability**. By default, Splunk ships with a special role, "can_delete" that has this capability (and no others). The admin role does not have this capability by default. Splunk recommends you create a special user that you log into when you intend to delete index data.

To use the delete operator, run a search that returns the events you want deleted. Make sure that this search **ONLY** returns events you want to delete, and no other events. Once you've confirmed that this is the data you want to delete, pipe that search to delete. Read more about how to *remove indexed data from Splunk* in the Managing Indexers and Clusters manual.

Note: The delete operator will trigger a roll of hot buckets to warm in the affected index(es).

Examples

Example 1: Delete events from the "insecure" index that contain strings that look like Social Security numbers.

```
index=insecure | regex _raw = "\d{3}-\d{2}-\d{4}" | delete
```

Example 2: Delete events from the "imap" index that contain the word "invalid"

```
index=imap invalid | delete
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the delete command.

delta

Synopsis

Computes the difference in field value between nearby results.

Syntax

```
delta (field [AS newfield]) [p=int]
```

Required arguments

field

Syntax: <fieldname>

Description: The name of a field to analyze.

Optional arguments

<newfield>

Syntax: <string>

Description: A rename for the `field` value.

p

Syntax: p=<int>

Description: If *newfield* is not specified, it defaults to `delta(field)`. If p is unspecified, the default = 1, meaning the immediate previous value is used.

Description

For each event where *field* is a number, the `delta` command computes the difference, in search order, between the event's value of the *field* and a previous event's value of *field* and writes this difference into *newfield*. If *newfield* is not specified, it defaults to `delta(field)`. If p is unspecified, it defaults to p=1, meaning that the immediate previous value is used. p=2 would mean that the value before the previous value is used, etc.

Note: The `delta` command works on the order of events. By default, the events we get for non-real-time searches are in reverse time order, from new events to old events; so, values ascending over time will show negative deltas. But, the `delta` could be applied after any sequence of commands, so there is no input order guaranteed.

Examples

Example 1

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

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_* action=purchase | top clientip | delta count p=1
```

Here, the purchase events (`action=purchase`) are piped into the `top` command to find the top ten users (`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. By default, this difference is saved in a field called `delta(count)`:

Overlay:

	clientip ↕	count ↕	percent ↕	delta(count) ↕
1	189.222.1.40	4	20.000000	
2	192.1.2.31	3	15.000000	-1
3	10.2.91.50	3	15.000000	0
4	10.32.1.49	2	10.000000	-1
5	10.192.1.31	2	10.000000	0
6	233.77.49.31	1	5.000000	-1
7	177.23.21.44	1	5.000000	0
8	10.32.1.43	1	5.000000	0
9	10.32.1.34	1	5.000000	0
10	10.32.1.30	1	5.000000	0

These results are formatted as a table because of the `top` command. Note that the first event does not have a `delta(count)` value.

Example 2

This example uses recent (October 18-25, 2010) earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains the source network (Src), ID (Eqid), version, date, location, magnitude, depth (km) and number of reporting stations (NST) for each earthquake over the last 7 days.

Download the text file, **M 2.5+ earthquakes, past 7 days**, save it as a CSV file, and upload it to Splunk. Splunk should extract the fields automatically. Note that you'll be seeing data from the 7 days previous to your download, so your results will vary from the ones displayed below.

Calculate the difference in time between each of the recent earthquakes in Northern California.

```
source="eqs7day-M1.csv" Region="Northern California" | delta _time AS
timeDeltaS p=1 | eval timeDeltaS=abs(timeDeltaS) | eval
timeDelta=tostring(timeDeltaS, "duration")
```

This example searches for earthquakes in Northern California (`Region="Northern California"`). Then it uses the `delta` command to calculate the difference in the timestamps (`_time`) between each earthquake and the one immediately before it. This change in time is renamed `timeDeltaS`.

This example also uses the `eval` command and `tostring()` function to reformat `timeDeltaS` as `HH:MM:SS`, so that it is more readable:

1	10/25/10 12:17:21.000 AM	nc,71478576,0,"Monday, October 25, 2010 07:17:21 UTC",38.7525,-122.7002,1.0,1.30, 8,"Northern California"	source=eqs7day-M1.csv	
2	10/24/10 10:39:18.000 PM	nc,71478531,1,"Monday, October 25, 2010 05:39:18 UTC",38.8307,-122.8770,2.0,2.80,30,"Northern California"	timeDeltaS=5883	timeDelta=01:38:03
3	10/24/10 9:52:05.000 PM	nc,71478521,0,"Monday, October 25, 2010 04:52:05 UTC",38.8310,-122.7977,1.0,2.20,12,"Northern California"	timeDeltaS=2633	timeDelta=00:47:13
4	10/24/10 9:15:45.000 PM	nc,71478506,0,"Monday, October 25, 2010 04:15:45 UTC",38.7743,-122.7233,1.3,2.60,16,"Northern California"	timeDeltaS=2180	timeDelta=00:36:20
5	10/24/10 8:19:37.000 PM	nc,71478496,0,"Monday, October 25, 2010 03:19:37 UTC",38.8123,-122.7690,1.1,0.30, 8,"Northern California"	timeDeltaS=3368	timeDelta=00:56:08

Here, you can see that: the difference between the first and second quake is almost 2 hours, the difference between the second and third is almost an hour later, etc.

Example 3

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Calculate the difference in time between consecutive transactions.

```
sourcetype=access_* | transaction JSESSIONID clientip
startswith="*signon*" 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, defines an event as the beginning of the transaction if it contains the string "signon" and the last event of the transaction if it contains the string "purchase".

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`.

This example also uses `eval` command to redefine `timeDelta` as its absolute value (`abs(timeDelta)`) and convert it to a more readable string format with the `tostring()` function.

1	10/24/10 11:44:40.000 PM	192.1.2.40 - - [24/Oct/2010:23:44:40] "POST /flower_store/j_signon_check HTTP/1.1" 302 309 "http://mystore.splunk.com/flower_store/enter_order_information.screen&JSESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3155 2516 192.1.2.40 - - [24/Oct/2010:23:44:40] "GET /flower_store/category.screen?category_id=GIFTS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-6&JSESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 1038 1872 JSESSIONID=SD5SL10FF8ADFF3 clientip=192.1.2.40 duration=0
2	10/24/10 11:35:21.000 PM	10.32.1.52 - - [24/Oct/2010:23:35:21] "POST /flower_store/j_signon_check HTTP/1.1" 302 309 "http://mystore.splunk.com/flower_store/enter_order_information.screen&JSESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3128 4728 10.32.1.52 - - [24/Oct/2010:23:35:21] "GET /flower_store/category.screen?category_id=CANDY HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-6&JSESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 178 1792 JSESSIONID=SD5SL10FF8ADFF3 clientip=10.32.1.52 duration=0 timeDelta=00:09:19
3	10/24/10 11:25:41.000 PM	10.192.1.34 - - [24/Oct/2010:23:25:41] "POST /flower_store/j_signon_check HTTP/1.1" 302 309 "http://mystore.splunk.com/flower_store/enter_order_information.screen&JSESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3211 3034 10.192.1.34 - - [24/Oct/2010:23:25:41] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-20&JSESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 2993 4217 JSESSIONID=SD5SL10FF8ADFF3 clientip=10.192.1.34 duration=0 timeDelta=00:09:40

You can see that: the difference between the first and second transactions is 9 minutes 19 seconds, the difference between the second and third transaction is 9 minutes 40 seconds, etc.

More examples

Example 1: Consider logs from a TV set top box (`sourcetype=tv`) that you can use to analyze broadcasting ratings, customer preferences, etc. Which channels do subscribers watch (`activity=view`) most and how long do they stay on those channels?

```
sourcetype=tv activity="View" | sort - _time | delta _time AS
timeDeltaS | eval timeDeltaS=abs(timeDeltaS) | stats sum(timeDeltaS) by
ChannelName
```

Example 2: Compute the difference between current value of count and the 3rd previous value of count and store the result in 'delta(count)'

```
... | delta count p=3
```

Example 3: For each event where 'count' exists, compute the difference between count and its previous value and store the result in 'countdiff'.

```
... | delta count AS countdiff
```

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

Synopsis

Returns the difference between two search results.

Syntax

```
diff [position1=int] [position2=int] [attribute=string] [diffheader=bool]  
[context=bool] [maxlen=int]
```

Optional arguments

position1

Datatype: <int>

Description: The position of a search result to compare to position2. By default, `position1=1` and refers to the first search result.

position2

Datatype: <int>

Description: The position of a search result, must be greater than position1. By 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. By default, `attribute=_raw`.

diffheader

Datatype: <bool>

Description: Specify whether to show (`diffheader=true`) or hide a header that explains the diff output. By default, `diffheader=false`.

context

Datatype: <bool>

Description: Specify whether to show (`context=true`) or hide context lines around the diff output. By default, `context=false`.

`maxlen`

Datatype: `<int>`

Description: Controls the maximum content in bytes diffed from the two events. By default, `maxlen=100000`, meaning 100KB; if `maxlen=0`, there is no limit.

Description

Compares two search results and returning the difference of the two. Which two search results are compared is specified by the two position values, which default to 1 and 2 (to compare the first two results). By default, the raw text (`_raw` attribute) of the two search results are compared, but other attributes can be specified with `attribute`. If `diffheader` is true, the traditional diff headers are created based on the source keys of the two events, it defaults to false. If `context` is true, context lines around the diff are shown; it defaults to false. If `maxlen` is provided, it controls the maximum content in bytes diffed from the two events. It defaults to 100000. If `maxlen=0`, there is no limit.

Examples

Example 1: Compare the "ip" values of the first and third search results.

```
... | diff pos1=1 pos2=3 attribute=ip
```

Example 2: Compare the 9th search results to the 10th.

```
... | diff position1=9 position2=10
```

See also

[set](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the diff command.

erex

Synopsis

Automatically extracts field values similar to the example values.

Syntax

```
erex [<field>] examples=<erex-examples> [counterexamples=<erex-examples>]  
[fromfield=<field>] [maxtrainers=<int>]
```

Required arguments

examples

Syntax: <erex-examples>

Description: A comma-separated list of example values for the information to be extracted and saved into a new field.

Optional arguments

counterexamples

Syntax: counterexamples=<erex-examples>

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 returned in an error message. That 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. Defaults to `_raw`.

maxtrainers

Syntax: maxtrainers=<int>

Description: The maximum number values to learn from. Must be between 1 and 1000. Defaults to 100.

Erex examples

<erex-examples>

Syntax: ""<string>(, <string>)*""

Description: A comma-separated list of example values.

Description

Example-based regular expression extraction. Automatically extracts field values from `fromfield` (defaults to `_raw`) that are similar to the `examples` (comma-separated list of example values) and puts them in `field`. If `field` is not specified, values are not extracted, but the resulting regular expression is generated and returned in an error message. That expression can then be used with the `rex` command for more efficient extraction. To learn the extraction rule for pulling out example values, it learns from at most `maxtrainers` (defaults to 100, must be between 1-1000).

Examples

Example 1: Extracts out values like "7/01" and "7/02", but not patterns like "99/2", putting extractions into the "monthday" attribute.

```
... | erex monthday examples="7/01, 07/02" counterexamples="99/2"
```

Example 2: Extracts out values like "7/01", putting them into the "monthday" attribute.

```
... | erex monthday examples="7/01"
```

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

Synopsis

Calculates an expression and puts the resulting value into a field.

Syntax

`eval eval-field=eval-expression`

Required arguments

`eval-field`

Syntax: <string>

Description: A field name for your evaluated value.

`eval-expression`

Syntax: <string>

Description: A combination of values, variables, operators, and functions that represent the value of your destination field. The syntax of the expression is checked before running the search, and an exception will be thrown for an invalid expression. For example, the result of an eval statement is not allowed to be boolean. If Splunk cannot evaluate the expression successfully at search-time for a given event, eval erases the value in the result field.

Operators

The following table lists the basic operations you can perform with eval. 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 may not produce valid results if the values are not numerical. When concatenating values, Splunk reads the values as strings (regardless of their value).

Type	Operators
Arithmetic	+ - * / %
Concatenation	.
Boolean	AND OR NOT XOR < > <= >= != = == LIKE

Functions

The eval command includes the following functions: `abs()`, `case()`, `ceil()`, `ceiling()`, `cidrmatch()`, `coalesce()`, `commands()`, `exact()`, `exp()`, `floor()`, `if()`, `ifnull()`, `isbool()`, `isint()`, `isnotnull()`, `isnull()`, `isnum()`, `isstr()`, `len()`, `like()`, `ln()`, `log()`, `lower()`, `ltrim()`, `match()`, `max()`,

`md5()`, `min()`, `mvappend()`, `mvcount()`, `mvindex()`, `mvfilter()`, `mvjoin()`, `mvrangle()`, `mvzip()`, `now()`, `null()`, `nullif()`, `pi()`, `pow()`, `random()`, `relative_time()`, `replace()`, `round()`, `rtrim()`, `searchmatch()`, `sigfig()`, `spath()`, `split()`, `sqrt()`, `strftime()`, `strptime()`, `substr()`, `time()`, `tonumber()`, `tostring()`, `trim()`, `typeof()`, `upper()`, `urldecode()`, `validate()`.

For **descriptions and examples** of each function, see ["Functions for eval and where"](#).

Description

Performs an evaluation of arbitrary expressions that can include mathematical, string, and boolean operations. The `eval` command requires that you specify a field name that takes the results of the expression you want to evaluate. If this destination field matches a field name that already exists, the values of the field are replaced by the results of the `eval` expression.

If you are using a search as an argument to the `eval` command and functions, you cannot use a saved search name; you have to pass a literal search string or a field that contains a literal search string (like the 'search' field extracted from `index=_audit` events).

You can use `eval` statements to define calculated fields. To do this, you set up the `eval` statement in `props.conf`. When you run a search, Splunk automatically evaluates the statements behind the scenes to create fields in a manner similar to that of search time field extraction. When you do this you no longer need to define the `eval` statement in a search string--you can just search on the resulting calculated field directly.

For more information see the [Calculated fields](#) section, below.

Examples

Example 1

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.

Example 2

This example uses recent (September 23-29, 2010) earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains the source network (Src), ID (Eqid), version, date, location, magnitude, depth (km) and number of reporting stations (NST) for each earthquake over the last 7 days.

Download the text file, **M 2.5+ earthquakes, past 7 days**, save it as a CSV file, and upload it to Splunk. Splunk should extract the fields automatically. Note that you'll be seeing data from the 7 days previous to your download, so your results will vary from the ones displayed below.

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).

Classify recent earthquakes based on their depth.

```
source=eqs7day-M1.csv | eval Description=case(Depth<=70, "Shallow",
Depth>70 AND Depth<=300, "Mid", Depth>300, "Deep") | table Datetime,
Region, Depth, 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 `eval` into the `table` command. This formats a table to display the timestamp of the earthquake, the region in which it occurred, the depth in kilometers of the quake, and the corresponding description assigned by the `eval` expression:

870 results over all time

« prev 1 2 3 4 5 6 7 8 9 10 next » | Options...

Results per page 10

Overlay: None

	Datetime ↕	Region ↕	Depth ↕	Description ↕
11	Wednesday, September 29, 2010 15:02:29 UTC	Southern Alaska	0.00	Shallow
12	Wednesday, September 29, 2010 14:57:31 UTC	Southern California	11.90	Shallow
13	Wednesday, September 29, 2010 14:10:46 UTC	Halmahera, Indonesia	237.40	Mid
14	Wednesday, September 29, 2010 13:51:23 UTC	Baja California, Mexico	3.40	Shallow
15	Wednesday, September 29, 2010 13:49:27 UTC	Central Alaska	136.30	Mid
16	Wednesday, September 29, 2010 13:10:16 UTC	Baja California, Mexico	9.30	Shallow
17	Wednesday, September 29, 2010 13:07:40 UTC	Southern California	4.00	Shallow
18	Wednesday, September 29, 2010 12:51:23 UTC	Central Alaska	0.00	Shallow
19	Wednesday, September 29, 2010 12:38:15 UTC	Arkansas	3.30	Shallow
20	Wednesday, September 29, 2010 12:11:17 UTC	Baja California, Mexico	10.00	Shallow

Example 3

This example is designed to use the sample dataset from **"Get the sample data into Splunk"** topic of the Splunk Tutorial, but it should work with any format of Apache Web access log. Download the data set and follow the instructions in that topic to upload it to Splunk. Then, run this search using the time range *Other > Yesterday*.

In this search, you're finding IP addresses and classifying the network they belong to.

```
sourcetype=access_* | eval network=if(cidrmatch("192.0.0.0/16",
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 `network` is given the value `local`. Otherwise, `network=other`.

The `eval` command does not do any special formatting to your results -- it just creates a new field which takes the value based on the eval expression. After you run this search, use the **fields sidebar** to add the `network` field to your results. 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 should look something like this:

9	9/30/10 11:59:00.000 PM	189.222.1.50 - - [30/Sep/2010:23:59:00] "GET /flower_store/product.screen?product_id=RP-SN-01 HTTP/1.1" 200 10893 "http://mystore.splunk.com/flower_store/category.screen?category_id=GIFTS&JSESSIONID=SD55L10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0-0.1.el4.centos Firefox/1.5.0.10" 3289 1339 network=other
10	9/30/10 11:58:46.000 PM	192.0.1.51 - - [30/Sep/2010:23:58:46] "GET /flower_store/category.screen?category_id=FLOWERS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-15&JSESSIONID=SD55L10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0-0.1.el4.centos Firefox/1.5.0.10" 4563 4843 network=local ← network
11	9/30/10 11:58:46.000 PM	192.0.1.51 - - [30/Sep/2010:23:58:46] "GET /flower_store/signoff.do HTTP/1.1" 200 9662 "http://mystore.splunk.com/flower_store/order.do&JSESSIONID=SD55L10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0-0.1.el4.centos Firefox/1.5.0.10" 1694 3552 network=local
12	9/30/10 11:58:10.000 PM	192.168.11.33 - - [30/Sep/2010:23:58:10] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-6&JSESSIONID=SD55L10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0-0.1.el4.centos Firefox/1.5.0.10" 3180 2291 network=other

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. (See Example 1)

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*.

Example 4

This example uses generated email data (`sourcetype=cisco_esa`). You should be able to run this example on any email data by replacing the `sourcetype=cisco_esa` with your data's `sourcetype` value and the `mailfrom` field with your data's email address field name (for example, it might be `To`, `From`, or `Cc`).

Use the email address field to extract the user's name and domain.

```
sourcetype="cisco_esa" mailfrom=* | eval
accountname=split(mailfrom,"@") | eval from_user=mvindex(accountname,0)
| eval from_domain=mvindex(accountname,-1) | table mailfrom, from_user,
from_domain
```

This example uses the `split()` function 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 example then uses `mvindex()` function to set `from_user` and `from_domain` to the first and second values of `accountname`, respectively.

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 following results table:

10 results in the last 60 minutes (from 7:59:00 AM to 8:59:43 AM on Thursday, October 7, 2010)

Options... Results per page 10

Overlay: None

	mailfrom ↕	from_user ↕	from_domain ↕
1	ariene98@yahoo.com	ariene98	yahoo.com
2	CostcoNews_F96829B393A8F1AE4A4D845417B24666@online.costco.com	CostcoNews_F96829B393A8F1AE4A4D845417B24666	online.costco.com
3	ESC1102793196416_1102110589836_839@in.constantcontact.com	ESC1102793196416_1102110589836_839	in.constantcontact.com
4	notification+zct1vf1@facebookmail.com	notification+zct1vf1	facebookmail.com
5	dom@yipnet.com	dom	yipnet.com
6	notification+zct1vf1@facebookmail.com	notification+zct1vf1	facebookmail.com
7	dom@yipnet.com	dom	yipnet.com
8	notification+orhhdzv1@facebookmail.com	notification+orhhdzv1	facebookmail.com
9	prvs=9451aaae2=automated@ecoupons.com	prvs=9451aaae2=automated	ecoupons.com
10	dom@yipnet.com	dom	yipnet.com

Note: This example is really not that practical. It 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 one (`accountname`) from a single value field (`mailfrom`).

Example 5

This example uses generated email data (`sourcetype=cisco_esa`). You should be able to run this example on any email data by replacing the `sourcetype=cisco_esa` with your data's `sourcetype` value and the `mailfrom` field with your data's email address field name (for example, it might be `To`, `From`, or `Cc`).

This example classifies where an email came from based on the email address's domain: `.com`, `.net`, and `.org` addresses are considered *local*, while anything else is considered *abroad*. (Of course, domains that are not `.com/.net/.org` or not necessarily from *abroad*.)

```
sourcetype="cisco_esa" mailfrom=* | eval accountname=split(mailfrom,"@")
| eval from_domain=mvindex(accountname,-1) | eval
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 Example 3. 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 `local`. If `from_domain` does not match, `location` is assigned `abroad`.

The `eval` results are then piped into the `stats` command to count the number of results for each `location` value and produce the following results table:

2 results in the last 60 minutes (from 8:27:00 AM to 9:27:08 AM on Thursday, October 7, 2010)

Results per page 10

Overlay: None

	domain_check :	count :
1	abroad	7
2	local	228

After you run the search, you can add the `mailfrom` and `location` fields to your events to see the classification inline with your events. If your search results contain these fields, they will look something like this:

24 fields | Pick fields

256 events in the last 60 minutes (from 8:28:00 AM to 9:28:37 AM on Thursday, October 7, 2010)

Results per page 10

51	10/7/10 9:23:36.000 AM	2010-10-7 9:23:36 2009 Info: MID 245292 ICID 744296 From: <slckdeals@slckdeals.net> location=local mailfrom=slckdeals@slckdeals.net
52	10/7/10 9:23:19.000 AM	2010-10-7 9:23:19 2009 Info: MID 245289 ICID 744291 From: <notification+206-a0oy@facebookmail.com> location=local mailfrom=notification+206-a0oy@facebookmail.com
53	10/7/10 9:23:08.000 AM	2010-10-7 9:23:8 2009 Info: MID 245288 ICID 744290 From: <ebay@ebay.com> location=local mailfrom=ebay@ebay.com
54	10/7/10 9:23:06.000 AM	2010-10-7 9:23:6 2009 Info: MID 245287 ICID 744290 From: <sentot@nafed.go.id> location=abroad mailfrom=sentot@nafed.go.id

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*.

Example 6

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 **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

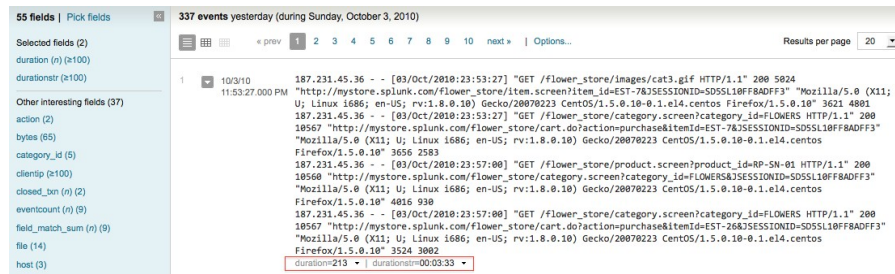
Reformat a numeric field measuring time in seconds into a more readable string format.

```
sourcetype=access_* | transaction clientip maxspan=10m | eval
durationstr=tostring(duration, "duration")
```

This example uses the `tostring()` function and the `duration` option to convert the `duration` of the transaction into a more readable string formatted as HH:MM:SS. The `duration` is the time between the first and last events in the transaction and is given in seconds.

The search defines a new field, `durationstr`, for the reformatted `duration` value.

After you run the search, you can use the Field picker to show the two fields inline with your events. If your search results contain these fields, they will look something like this:



More examples

Example A: Set velocity to distance / time.

```
... | eval velocity=distance/time
```

Example B: Set status to OK if error is 200; otherwise, Error.

```
... | eval status = if(error == 200, "OK", "Error")
```

Example C: Set lowuser to the lowercase version of username.

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

Example D: 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)
```

Example E: Set status to some simple http error codes.

```
... | eval error_msg = case(error == 404, "Not found", error == 500, "Internal Server Error", error == 200, "OK")
```

Example F: Set full_name to the concatenation of first_name, a space, and last_name.

```
... | eval full_name = first_name." ".last_nameSearch
```

Example G: Display timechart of the avg of cpu_seconds by processor rounded to 2 decimal places.

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

Example H: Convert a numeric field value to a string with commas and 2 decimal places. If the original value of x is 1000000, this returns x as 1,000,000.

```
... | eval x=toString(x, "commas")
```

Calculated fields

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 [Example 4](#), above, to:

```
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 setting calculated fields up in `props.conf`, see "Define calculated fields" in the Knowledge Manager Manual.

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the eval command.

eventcount

Synopsis

Returns the number of events in an index.

Syntax

```
eventcount [index=<string>] [summarize=<bool>]
```

Optional arguments

index

Syntax: index=<string>

Description: The name of the index to count events, instead of the default index.

summarize

Syntax: summarize=<bool>

Description: Specifies whether or not to summarize eventcounts.

Examples

Example 1: Gives event count by each index/server pair.

```
| eventcount summarize=false index=*
```

Example 2: Displays event count over all search peers.

```
| eventcount summarize=true
```

Example 3: Return the number of events in the '_internal' index.

```
| eventcount index=_internal
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the eventcount command.

eventstats

Synopsis

Adds summary statistics to all search results.

Syntax

eventstats [allnum=<bool>] <stats-agg-term>* [<by clause>]

Required arguments

<stats-agg-term>

Syntax: <stats-func>(<evaluated-field> | <wc-field>) [AS <wc-field>]

Description: A statistical specifier optionally renamed to a new field name. The specifier can be by an aggregation function applied to a field or set of fields or an aggregation function applied to an arbitrary eval expression.

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 is false.)

<by clause>

Syntax: by <field-list>

Description: The name of one or more fields to group by.

Stats functions options

stats-function

Syntax: avg() | c() | count() | dc() | distinct_count() | first() | last() | list() | max() | median() | min() | mode() | p<in>() | perc<int>() | per_day() | per_hour() | per_minute() | per_second() | range() | stdev() | stdevp() | sum() | sumsq() | values() | var() | varp()

Description: Functions used with the stats command. Each time you invoke the `stats` command, you can use more than one function; however, you can only use one `by clause`. For a list of stats functions with descriptions and examples, see "[Functions for stats, chart, and timechart](#)".

Description

Generate summary statistics of all existing fields in your search results and save them as values in new fields. Specify a new field name for the statistics results by using the **as** argument. If you don't specify a new field name, the default field name is the statistical operator and the field it operated on (for example: **stat-operator(field)**). Just like the `stats` command except that aggregation results are added inline to each event and only the aggregations that are pertinent to that event. The **allnum** option has the same meaning as that option in the `stats` command.

Examples

Example 1: Compute the overall average duration and add 'avgdur' as a new field to each event where the 'duration' field exists

```
... | 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`.

```
... | 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.

```
eventtype="error" | eventstats avg(foo) as avg | where foo>avg
```

See also

[stats](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the eventstats command.

extract (kv)

Synopsis

Extracts field-value pairs from search results.

Syntax

```
extract | kv <extract-opt>* <extractor-name>*
```

Required arguments

<extract-opt>

Syntax: auto=<bool> | clean_keys=<bool> | kvdelim=<string> | limit=<int> | maxchars=<int> | mv_add=<bool> | pairdelim=<string> | reload=<bool> | segment=<bool>

Description: Options for defining the extraction.

<extractor-name>

Syntax: <string>

Description: A stanza that can be found in transforms.conf. This is used when props.conf did not explicitly cause an extraction for this source, sourcetype, or host.

Extract options

auto

Syntax: auto=<bool>

Description: Specifies whether to perform automatic "=" based extraction. Defaults to true.

clean_keys

Syntax: clean_keys=<bool>

Description: Specifies whether to clean keys. Overrides CLEAN_KEYS from transforms.conf.

kvdelim

Syntax: kvdelim=<string>

Description: Specify 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. Defaults to 50.

maxchars

Syntax: maxchars=<int>

Description: Specifies how many characters to look into the event. Defaults to 10240.

mv_add

Syntax: mv_add=<bool>

Description: Specifies whether to create multivalued fields. Overrides MV_ADD from transforms.conf.

pairdelim

Syntax: pair=<string>

Description: Specify 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 props.conf and transforms.conf. Defaults to false.

segment

Syntax: segment=<bool>

Description: Specifies whether to note the locations of key/value pairs with the results. Defaults to false.

Description

Forces field-value extraction on the result set.

Examples

Example 1: Extract field/value pairs that are delimited by "|;", and values of fields that are delimited by "=: ". Note that the delimiters are individual characters. So in this example the "=" or ":" will be used to delimit the key value. Similarly, a "|" or ";" will be used to delimit against the pair itself.

```
... | extract pairdelim="|;", kvdelim="=: ", auto=f
```

Example 2: Extract field/value pairs and reload field extraction settings from disk.

```
... | extract reload=true
```

Example 3: Extract field/value pairs that are defined in the transforms.conf stanza 'access-extractions'.

```
... | extract access-extractions
```

See also

[kvform](#), [multikv](#), [rex](#), [xmlkv](#),

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the extract command.

fieldformat

The `fieldformat` command enables you to use eval expressions to change the format of a field value when the results render.

Note: This does not apply when exporting data (to a csv file, for example) because export retains the original data format rather than the rendered format. There is no option to the Splunk Web export interface to render fields.

Synopsis

Expresses how to render a field at output time without changing the underlying value.

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. For more information, see the [eval command reference](#) and the [list of eval functions](#).

Examples

Example 1: Specify that the start_time should be rendered by taking the value of start_time (assuming it is an epoch number) and rendering it to display just the hours minutes and seconds corresponding that epoch time.

```
... | fieldformat start_time = strftime(start_time, "%H:%M:%S")
```

See also

[eval](#), [where](#)

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the fieldformat command.

fields

Synopsis

Keeps or removes fields from search results.

Syntax

fields [+|-] <wc-field-list>

Required arguments

<wc-field-list>

Syntax: <string>, ...

Description: Comma-delimited list of fields to keep (+) or remove (-); can include wildcards.

Description

Keeps (+) or removes (-) fields based on the field list criteria. If + is specified, only the fields that match one of the fields in the list are kept. If - is specified, only the fields that match one of the fields in the list are removed.

Without either + or -, it is the equivalent to calling with + and adding `_*` to the list -- that is, "fields x, y" is the same as "fields + x, y, `_*`".

Important: The leading underscore is reserved for all internal Splunk field names, such as `_raw` and `_time`. By default, internal fields `_raw` and `_time` are included in output. The `fields` command does not remove internal fields unless explicitly specified with:

```
... | fields - _*
```

or more explicitly, with:

```
... | fields - _raw,_time
```

Note: DO NOT remove the `_time` field when you pipe results to statistical commands.

Examples

Example 1: Remove the "host" and "ip" fields.

```
... | fields - host, ip
```

Example 2: Keep *only* the "host" and "ip" fields, and display them in the order: "host", "ip". Note that this also removes the internal fields, which begin with an underscore (such as `_time`).

```
... | fields host, ip | fields - _*
```

Example 3: 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

Synopsis

Generates summary information for all or a subset of the fields.

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 is 100.

wc-field-list

Syntax:

Description: A field, or list of fields, including wildcarded fields.

Examples

Example 1: Return summaries for all fields.

```
... | fieldsummary
```

Example 2: Returns summaries for only fields that start with date_ and return only the top 10 values for each field.

```
... | fieldsummary maxvals=10 date_*
```

See also

af, 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

Synopsis

Replace null values with last non-null value.

Syntax

```
filldown <wc-field-list>
```

Description

Replace null values with the last non-null value for a field or set of fields. If no list of fields is given, filldown will be applied to all fields. If there were not any previous values for a field, it will be left blank (NULL).

Examples

Example 1: Filldown null values values for all fields.

```
... | filldown
```

Example 2: Filldown null values for the count field only.

```
... | filldown count
```

Example 3: Filldown null values for the count field and any field that starts with 'score'.

```
... | filldown count score*
```

See also

[fillnull](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the filldown command.

fillnull

Synopsis

Replaces null values with a specified value.

Syntax

```
fillnull [value=string] <field-list>
```

Required arguments

field-list

Syntax: <field>...

Description: One or more fields, delimited with a space. If not specified, fillnull is applied to all fields.

Optional arguments

value

Datatype: <string>

Description: Replaces null values with a user specified value (default 0)

Description

Replaces null values with a user specified value (default 0). Null values are those missing in a particular result, but present for some other result. If a field-list is

provided, fillnull is applied to only fields in the given list (including any fields that does not exist at all). Otherwise, applies to all existing fields.

Examples

Example 1: For the current search results, fill all empty fields with NULL.

```
... | fillnull value=NULL
```

Example 2: For the current search results, fill all empty field values of "foo" and "bar" with NULL.

```
... | fillnull value=NULL foo bar
```

Example 3: For the current search results, fill all empty fields with zero.

```
... | 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

[streamstats](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the fillnull command.

findtypes

Synopsis

Generates suggested eventtypes.

Syntax

```
findtypes max=<int> [notcovered] [useraw]
```

Required arguments

max

Datatype: <int>

Description: The maximum number of events to return. Defaults to 10.

Optional arguments

notcovered

Description: If this keyword is used, findtypes returns only event types that are not already covered.

useraw

Description: If this keyword is used, findtypes uses phrases in the `_raw` text of events to generate event types.

Description

The findtypes command takes the results of a search and produces a list of promising searches that may be used as event types. At most, 5000 events are analyzed for discovering 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

Synopsis

Replaces `attr` with higher-level grouping, such as replacing filenames with directories.

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 happens via tokenizing the `attr` value on the `sep` separator value.

`sep`

Syntax: `sep=<string>`

Description: Used to construct output field names when multiple data series are used in conjunctions with a split-by field. Defaults to `::`

`size`

Syntax: `size=<string>`

Description: Defaults to `totalCount`.

`minfolders`

Syntax: `minfolders=<int>`

Description: Set the minimum number of folders to group. Defaults to 2.

`maxfolders`

Syntax: `maxfolders=<int>`

Description: Set the maximum number of folders to group. Defaults to 20.

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 happens via tokenizing the `attr` value on the `sep` separator value. For example, it can group

search results, such as those used on the Splunk homepage to list hierarchical buckets (e.g. directories or categories). Rather than listing 200 sources on the Splunk homepage, `folderize` breaks the source strings by a separator (e.g. `/`), and determines if looking at just directories results in the number of results requested. The default `sep` separator is `:`; the default size attribute is `totalcount`; the default `minfolders` is 2; and the default `maxfolders` is 20.

Examples

Example 1: Example usage

```
| metadata type=sources | folderize maxfolders=20 attr=source sep="/" |  
sort totalcount d
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `folderize` command.

format

Synopsis

Takes the results of a **subsearch** and formats them into a single result.

Syntax

```
format ["<string>" "<string>" "<string>" "<string>" "<string>" "<string>"]
```

Optional arguments

<string>

Syntax: "<string>"

Description: These six optional string arguments correspond to: ["<row prefix>" "<column prefix>" "<column separator>" "<column end>" "<row separator>" "<row end>"]. By default, when you don't specify any strings, the format output defaults to: " (" (" "AND" ") " "OR" ") "

Description

Used implicitly by subsearches, to take the search results of a subsearch and return a single result that is a query built from the input search results.

Examples

Example 1: Get top 2 results and create a search from their host, source and sourcetype, resulting in a single search result with a "query" field: `query=(("host::mylaptop" AND "source::syslog.log" AND "sourcetype::syslog") OR ("host::bobsllaptop" AND "source::bob-syslog.log" AND "sourcetype::syslog"))`

```
... | head 2 | fields source, sourcetype, host | format
```

Example 2: Increase the maximum number of events from the default to 2000 for a subsearch to use in generating a search.

In `limits.conf`:

```
[format]
maxresults = 2000
```

and in the subsearch:

```
... | head 2 | fields source, sourcetype, host | format maxresults=2000
```

See also

[search](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `format` command.

gauge

The gauge chart types enable you to see a single numerical value mapped against a range of colors that may have particular business meaning or business logic. As the value changes over time, the gauge marker changes position within this range.

The `gauge` command enables you to indicate the field whose value will be tracked by the gauge chart. You can define the overall numerical range represented by the gauge, and you can define the size of the colored bands within that range. If you want to use the color bands, you can add four "range values" to the search string that indicate the beginning and end of the range as

well as the relative sizes of the color bands within it.

Read more about using the gauge command with the gauge chart type in the Chart Gallery's subtopic about Gauge.

Synopsis

Transforms results into a format suitable for display by the Gauge chart types.

Syntax

gauge [<num>|<field>]...

Arguments

num

Description: At least one real number, delimited by a space.

field

Description: The name of a field. The values of the field in the first input row is used.

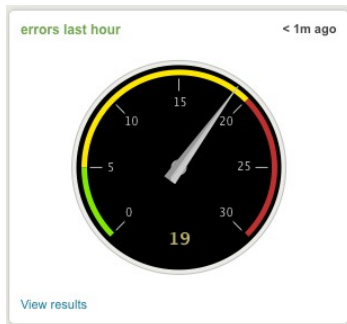
Description

Each argument is either a real number or the name of a field. The first argument is the gauge value and is required. Each argument after that is optional and defines the range for different sections of the gauge. If you don't provide at least two range numbers, the gauge will start at 0 and end at 100. If an argument is a field name, the value of that field in the first input row is used. This command is implemented as an external python script.

Examples

Example 1: Count the number of events and display the count on a gauge with 4 regions, (0-750, 750-1000, 1000-1250,1250-1500).

```
index=_internal | stats count as myCount | gauge myCount 750 1000 1250 1500
```



There are three types of gauges that you can choose from: radial, filler, and marker. You can see more examples of gauges in the Chart Gallery's subtopic about Gauge.

See also

[eval](#), [stats](#)

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the gauge command.

gentimes

Synopsis

Generates time range results. This command is useful in conjunction with the [map](#) command.

Syntax

```
gentimes start=<timestamp> [end=<timestamp>] [<increment>]
```

Required arguments

start

Syntax: start=<timestamp>

Description: Specify as start time.

<timestamp>

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

Description: Indicate the time, for example: 10/1/07:12:34:56 (for October 1, 2007 12:34:56) or -5 (five days ago).

Optional arguments

end

Syntax: end=<timestamp>

Description: Specify and end time.

<increment>

Syntax: increment=<int>(s|m|h|d)

Description: Specify a time period to increment from the start time to the end time.

Examples

Example 1: All HOURLY time ranges from oct 1 till oct 5

```
| gentimes start=10/1/07 end=10/5/07 increment=1h
```

Example 2: All daily time ranges from 30 days ago until 27 days ago

```
| gentimes start=-30 end=-27
```

Example 3: All daily time ranges from oct 1 till oct 5

```
| gentimes start=10/1/07 end=10/5/07
```

Example 4: All daily time ranges from oct 25 till today

```
| gentimes start=10/25/07
```

See also

[map](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the gentimes command.

head

Synopsis

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 realtime search.

Syntax

head [<N> | <eval-expression>] [limit=<int>] [null=<bool>] [keeplast=<bool>]

Optional arguments

eval-expression

Syntax: <eval-math-exp> | <eval-concat-exp> | <eval-compare-exp> | <eval-bool-exp> | <eval-function-call>

Description: A valid eval expression that evaluates to a Boolean. Splunk returns results until this expression evaluates to false. For more information, see the [Functions for eval](#).

keeplast

Syntax: keeplast=<bool>

Description: Controls whether or not to keep the last event, which caused the eval expression to evaluate to false (or NULL).

limit

Syntax: limit=<int>

Description: Another way to specify the number of results to return. Defaults to 10.

<N>

Syntax: <int>

Description: The number of results to return. If none is specified, Defaults to 10.

null

Syntax: null=<bool>

Description: If instead of specifying a number N, you use a boolean eval expression, this specifies how a null result should be 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. So, null=true means that the head command continues if it gets a null result, and null=false means the command stops if that happens.

Description

Returns the first n results, or 10 if no integer is specified. New for 4.0, can provide a boolean eval expression, in which case we return events until that expression evaluates to false.

Examples

Example 1: Return the first 20 results.

```
... | head 20
```

Example 2: Return events until the time span of the data is >= 100 seconds

```
... | streamstats range(_time) as timerange | 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

Synopsis

Causes `ui` to highlight specified terms.

Syntax

highlight <string>+

Required arguments

<string>

Syntax: <string>,...

Description: Comma-separated list of keywords to highlight in results.

Description

Causes the strings provided to be highlighted by **Splunk Web**.

Examples

Example 1: Highlight the terms "login" and "logout".

```
... | highlight login,logout
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the highlight command.

history

Synopsis

Returns a history of searches formatted as an events list or as a table.

Syntax

```
history [events=<bool>]
```

Arguments

events

Syntax: events= T | F

Description: Specify whether to return the search history as an events list (T) or as a table (F). Defaults to F.

Examples

Example 1: Return a table of the search history.

```
... | history
```

See also

[search](#)

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the history command.

iconify

Synopsis

Causes **Splunk Web** to make a unique icon for each value of the fields listed.

Syntax

iconify <field-list>

Required arguments

field-list

Syntax: <field>...

Description: Comma or space-delimited list of non-wildcarded fields.

Description

Displays a different icon for each field's unique value. If multiple fields are listed, the UI displays a different icon for each unique combination of the field values.

Examples

Example 1: Displays an different icon for each eventtype.

```
... | iconify eventtype
```

Example 2: Displays an different icon for unique pairs of `clientip` and `method` values.

```
... | iconify clientip method
```

Here's how Splunk displays the results in your **Events List**:

7	5/30/10 11:55:27.000 PM		10.31.10.21 - - [30/May/2010:23:55:27] "GET / HTTP/1.1" 200 9619 "-" "Feedfetcher-Google; (<http://www.google.com/feedfetcher.html; feed-id=18085321451258128740)" "10.31.10.21.1269586527948788" clientip=10.31.10.21 method=GET
8	5/30/10 11:55:01.000 PM		10.135.143.166 - - [30/May/2010:23:55:01] "GET /view/SP-CAAACGY HTTP/1.1" 200 11729 "http://5207724e63.example.org/" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.2.2) Gecko/20100316 YFF35 Firefox/3.6.2" "216.145.54.158.1265888041846647" clientip=10.135.143.166 method=GET
9	5/30/10 11:54:20.000 PM		10.135.143.166 - - [30/May/2010:23:54:20] "GET /download HTTP/1.1" 200 7501 "-" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.2.2) Gecko/20100316 YFF35 Firefox/3.6.2" "216.145.54.158.1265888041846647" clientip=10.135.143.166 method=GET

See also

[highlight](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the iconify command.

input

Synopsis

Adds or disables sources from being processed by Splunk.

Syntax

input (add|remove) [sourcetype=*string*] [index=*string*] [*string*=*string*]*

Optional arguments

sourcetype

Datatype: <string>

Description: Adds or removes (disables) sources from being processed by splunk, enabling or disabling inputs in inputs.conf, with optional sourcetype and index settings.

index

Datatype: <string>

Description: Adds or removes (disables) sources from being processed by splunk, enabling or disabling inputs in inputs.conf, with optional sourcetype and index settings.

Description

Adds or removes (disables) sources from being processed by splunk, enabling or disabling inputs in inputs.conf, with optional sourcetype and index settings. Any additional attribute=values are set added to inputs.conf. Changes are logs to \$splunk_home/var/log/splunk/inputs.log. Generally to be used in conjunction with the `crawl` command.

Examples

Example 1: Remove all csv files that are currently being processed

```
| crawl | search source=*csv | input remove
```

Example 2: Add all sources found in bob's home directory to the 'preview' index with sourcetype=text, setting custom user fields 'owner' and 'name'

```
| crawl root=/home/bob/txt | input add index=preview sourcetype=text  
owner=bob name="my nightly crawl"
```

Example 3: Add each source found by crawl in the default index with automatic source classification (sourcetype)

```
| crawl | input add
```

See also

[crawl](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the input command.

inputcsv

Synopsis

Loads search results from the specified `csv` file.

Syntax

```
inputcsv [append=<bool>] [start=<int>] [max=<int>] [events=<bool>] <filename>
```

Required arguments

filename

Syntax: <filename>

Description: Specify the name of the CSV file, located in
\$SPLUNK_HOME/var/run/splunk.

Optional arguments

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). Defaults to 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. Defaults to 1000000000.

start

Syntax: start=<int>

Description: Controls the 0-based offset of the first event to be read. Defaults to 0.

Description

Populates the results data structure using the given csv file, which is not modified. The filename must refer to a relative path in \$SPLUNK_HOME/var/run/splunk and if the specified file does not exist and the filename did not have an extension, then filename with a `.csv` extension is assumed.

Note: If you run into an issue with inputcsv resulting in an error, make sure that your CSV file ends with a BLANK LINE.

Examples

Example 1: Read in results from the CSV file:

"\$SPLUNK_HOME/var/run/splunk/all.csv", keep any that contain the string "error", and save the results to the file:

"\$SPLUNK_HOME/var/run/splunk/error.csv"

```
| inputcsv all.csv | search error | outputcsv errors.csv
```

Example 2: Read in events 101 to 600 from either file 'bar' (if exists) or 'bar.csv'.

```
| inputcsv start=100 max=500 bar
```

Example 3: Read in events from the CSV file:

"\$SPLUNK_HOME/var/run/splunk/foo.csv".

```
| inputcsv foo.csv
```

See also

[outputcsv](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `inputcsv` command.

inputlookup

Synopsis

Loads search results from a specified static lookup table.

Syntax

```
inputlookup [append=<bool>] [start=<int>] [max=<int>] (<filename> |  
<tablename>)
```

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, Splunk will display a warning message (but it

won't cause a syntax error).

<tablename>

Syntax: <string>

Description: The name of the lookup table as specified by a stanza name in transforms.conf.

Optional arguments

append

Syntax: append=<bool>

Description: If set to true, the data 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.

Description

Reads in lookup table as specified by a filename (must end with .csv or .csv.gz) or a table name (as specified by a stanza name in transforms.conf). If 'append' is set to true (false by default), the data from the lookup file is appended to the current set of results rather than replacing it.

Examples

Example 1: Read in "usertogroup" lookup table (as specified in transforms.conf).

```
| inputlookup usertogroup
```

Example 2: Same as example2 except that the data from the lookup table is appended to any current results.

```
| inputlookup append=t usertogroup
```

Example 3: Read in "users.csv" lookup file (under \$SPLUNK_HOME/etc/system/lookups or \$SPLUNK_HOME/etc/apps/*/lookups).

```
| inputlookup users.csv
```

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

Synopsis

Extracts location information from `ip` addresses.

Syntax

```
iplocation [maxinputs=<int>]
```

Optional arguments

maxinputs

Syntax: maxinputs=<int>

Description: Specifies how many of the top results are passed to the script.

Description

Finds IPs in `_raw` and looks up the ip location using the `hostip.info` database ips are extracted as `ip1,ip2` etc. and Cities and Countries are likewise extracted.

Examples

Example 1: Add location information (based on IP address).


```
... | iplocation
```

Example 2: Search for client errors in Web access events, add the location information, and return a table of the IP address, City and Country for each client error.

```
404 host="webserver1" | head 20 | iplocation | table clientip, City, Country
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `iplocation` command.

join

A join is used to combine the results of a search and subsearch if specified fields are common to each. You can also join a table to itself using the [selfjoin](#) command.

Synopsis

SQL-like joining of results from the main results pipeline with the results from the subpipeline.

Syntax

```
join [join-options]* <field-list> [ subsearch ]
```

Required arguments

subsearch

Description: A search pipeline. Read more about how subsearches work in the *Search manual*.

Optional arguments

field-list

Syntax: <field>, ...

Description: Specify the exact fields to use for the join. If none are specified, uses all fields that are common to both result sets.

join-options

Syntax: type=(inner|outer|left) | usetime=<bool> | earlier=<bool> | overwrite=<bool> | max=<int>

Description: Options to the join command.

Join options

type

Syntax: type=inner | outer | left

Description: Indicates the type of join to perform. Basically, the difference between an `inner` and a `left` (or `outer`) join is how they treat events in the main pipeline that do not match any in the subpipeline. In both cases, events that match are joined. The results of an `inner` join will not include any events with no matches. A `left` (or `outer`) join does not require each event to have matching field values; and the joined result retains each event?even if there is no match with any rows of the subsearch. Defaults to `inner`.

usetime

Syntax: usetime=<bool>

Description: Indicates whether to limit matches to sub-results that are earlier or later than the main result to join with. Defaults to `false`.

earlier

Syntax: earlier=<bool>

Description: If `usetime=true`, specify whether to join with matches that are earlier (`true`) or later (`false`) than the main result. Defaults to `true`.

overwrite

Syntax: overwrite=<bool>

Description: Indicates if fields from the sub results should overwrite those from the main result if they have the same field name. Defaults to `true`.

max

Syntax: max=<int>

Description: Indicates the maximum number of sub-results each main result can join with. If `max=0`, means no limit. Defaults to `1`.

Description

Traditional join command that joins results from the main results pipeline with the results from the search pipeline provided as the last argument. Optionally specifies the exact fields to join on. If no fields specified, will use all fields that are common to both result sets.

Examples

Example 1: Joins previous result set with results from 'search foo', on the id field.

```
... | join id [search foo]
```

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

Synopsis

Performs k-means clustering on selected fields.

Syntax

```
kmeans [kmeans-options]* <field-list>
```

Required arguments

field-list

Syntax: <field>, ...

Description: Specify the exact fields to use for the join. If none are specified, uses all fields that are common to both result sets.

Optional arguments

kmeans-options

Syntax: <reps>|<iters>|<tol>|<k>|<cnumfield>|<distype>

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. Defaults to 10.

iters

Syntax: maxiters=<int>

Description: Specify the maximum number of iterations allowed before failing to converge. Defaults to 10000.

tol

Syntax: tol=<num>

Description: Specify the algorithm convergence tolerance. Defaults to 0.

k

Syntax: k=<int>|<int>-<int>

Description: Specify the number of initial clusters to use. This value can be expressed as a range; in this case, each value in the range will be used once and the summary data given. Defaults to 2.

cnumfield

Syntax: cfield=<field>

Description: Names the field for the cluster number for each event. Defaults to CLUSTERNUM.

distype

Syntax: dt=l1|l1norm|cityblock|cb|l2|l2norm|sq|sqeuclidean|cos|cosine

Description: Specify the distance metric to use. L1/L1NORM is equivalent to CITYBLOCK. L2NORM is equivalent to SQUEULIDEAN. Defaults to L2NORM.

Description

Performs k-means clustering on select fields (or all numerical fields if empty). Events in the same cluster will be moved next to each other. Optionally the cluster number for each event is displayed.

Examples

Example 1: Group search results into 4 clusters based on the values of the "date_hour" and "date_minute" fields.

```
... | kmeans k=4 date_hour date_minute
```

Example 2: Group results into 2 clusters based on the values of all numerical fields.

... | 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

Synopsis

Extracts values from search results, using a form template.

Syntax

kvform [form=<string>] [field=<field>]

Optional arguments

form

Syntax: form=<string>

Description: Specify a .form file located in

\$SPLUNK_HOME/etc/apps/.../form.

field

Syntax: <field>

Description: The name of the field to extract. Defaults to `sourcetype`.

Description

Extracts key/value pairs from events based on a form template that describes how to extract the values. If `form` is specified, it uses an installed `form.form` file found in the Splunk configuration form directory. For example, if

`form=sales_order`, would look for a `sales_order.form` file in

\$SPLUNK_HOME/etc/apps/.../form. All the events processed would be matched against that form, trying to extract values.

If no FORM is specified, then the `field` value determines the name of the field to extract. For example, if `field=error_code`, then an event that has an `error_code=404`, would be matched against a `404.form` file.

The default value for `field` is `sourcetype`, thus by default the `kvform` command will look for `SOURCETYPE.form` files to extract values.

A `.form` file is essentially a text file of all static parts of a form. It may be interspersed with named references to regular expressions of the type found in `transforms.conf`. An example `.form` file might look like this:

```
Students Name: [[string:student_name]]
Age: [[int:age]] Zip: [[int:zip]]
```

Examples

Example 1: Extract values from "eventtype.form" if the file exists.

```
... | 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

Synopsis

Loads events or results of a previously completed search job.

Syntax

```
loadjob (<sid>|<savedsearch-opt>) [<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>:<application-string>:<search-name-string>"

Description: The unique identifier of a savedsearch whose artifacts need to be loaded. A savedsearch is uniquely identified by the triplet {user, application, savedsearch name}, for example:

savedsearch="admin:search:my saved search"

Optional arguments

result-event

Syntax: events=<bool>

Description: Controls whether to load the events or the results of a job. Defaults to false (loads results).

delegate

Syntax: job_delegate=<string>

Description: When specifying a savedsearch, this option selects jobs that were started by the given user. Defaults to scheduler.

artifact-offset

Syntax: artifact_offset=<int>

Description: If multiple artifacts are found, this specifies which of those should be loaded. Artifacts are sorted in descending order based on the time that they were started. Defaults to 0.

ignore_running

Syntax: ignore_running=<bool>

Description: Specify whether to ignore matching artifacts whose search is still running. Defaults to true.

Description

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.

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=t
```

See also

[inputcsv](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the loadjob command.

localize

Synopsis

Returns a list of time ranges in which the search results were found.

Syntax

localize [<maxpause>] [<timeafter>] <timebefore>

Required arguments

timebefore

Syntax: timebefore=<int>(s|m|h|d)

Description: Specify the amount of time to subtract from starttime (expand the time region backwards in time). Defaults to 30s.

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. Defaults to 1m.

timeafter

Syntax: maxpause=<int>(s|m|h|d)

Description: Specify the amount of time to add to endtime (expand the time region forward in time). Defaults to 30s.

Description

Generates a list of time contiguous event regions defined as: a period of time in which consecutive events are separated by at most 'maxpause' time. The found regions can be expanded using the 'timeafter' and 'timebefore' modifiers to expand the range after/before the last/first event in the region respectively. The Regions are return in time descending order, just as search results (time of region is start time). The regions discovered by localize are meant to be feed into the `map` command, which will use a different region for each iteration. Localize also reports: (a) number of events in the range, (b) range duration in seconds and (c) region density defined as (#of events in range) divided by (range duration) - events per second.

Examples

Example 1: Search the time range of each previous result for "failure".

```
... | localize maxpause=5m | map search="search failure  
starttimeu=$starttime$ endtimeu=$endtime$"
```

Example 2: As an example, searching for "error" and then calling localize finds good regions around where error occurs, and passes each on to the search inside of the map command, so that each iteration works with a specific timerange to find promising 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

Synopsis

Prevents subsequent commands from being executed on remote peers.

Syntax

`localop`

Description

Prevents subsequent commands from being executed on remote peers, i.e. forces subsequent commands to be part of the reduce step.

Examples

Example 1: The `iplocation` command in this case will never be run on remote peers. All events from remote peers from the initial search for the terms `FOO` and `BAR` will be forwarded to the search head where the `iplocation` command will be run.

```
FOO BAR | localop | iplocation
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `localop` command.

lookup

Use the `lookup` command to manually invoke field lookups from a lookup table that you've defined in `transforms.conf`. For more information, see "Lookup fields from external data sources," in the Knowledge Manager manual.

Synopsis

Explicitly invokes field value lookups.

Syntax

```
lookup [local=<bool>] [update=<bool>] <lookup-table-name> ( <lookup-field> [AS  
<local-field>] ) ( OUTPUT | OUTPUTNEW <lookup-destfield> [AS  
<local-destfield>] )
```

Required arguments

<lookup-table-name>

Syntax: <string>

Description: Refers to a stanza name in transforms.conf. This stanza specifies the location of the lookup table file.

Optional arguments

local

Syntax: local=<bool>

Description: If the 'local' option is set to true, it will ensure that the lookup is only done locally and not on any remote peers.

update

Syntax: update=<bool>

Description: If the lookup table is updated on disk while the search is running, real-time searches will reflect the update while non-real-time search will not. If you want to automatically update lookups for real-time searches, specify update=true (this also implies that local=true). Defaults to false.

<local-destfield>

Syntax: <string>

Description: Refers to the field in the local event, defaults to the value of <lookup-destfield>.

<local-field>

Syntax: <string>

Description: Refers to the field in the local event, defaults to the value of <lookup-field>.

<lookup-destfield>

Syntax: <string>

Description: Refers to a field in the lookup table to be copied to the local event.

<lookup-field>

Syntax: <string>

Description: Refers to a field in the lookup table to match to the local event.

Description

Use the lookup command to invoke field value lookups manually.

If an OUTPUT clause is not specified, all fields in the lookup table that are not specified as a *lookup* will be used as output fields. If OUTPUT is specified, the output lookup fields will overwrite existing fields. If OUTPUTNEW is specified, the lookup will not be performed for events in which the output fields already exist.

Examples

Example 1: There is a lookup table specified in a stanza name 'usertogroup' in transform.conf. This lookup table contains (at least) two fields, 'user' and 'group'. For each event, we look up the value of the field 'local_user' in the table and for any entries that matches, the value of the 'group' field in the lookup table will be written to the field 'user_group' in the event.

```
... | lookup usertogroup user as local_user OUTPUT group as user_group
```

Optimizing your lookup search

If you're using the lookup command in the same pipeline as a reporting command, do the lookup after the reporting command. For example, run:

```
sourcetype=access_* | stats count by status | lookup status_desc status  
OUTPUT description
```

instead of:

```
sourcetype=access_* | lookup status_desc status OUTPUT 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.

See also

[appendcols](#), [inputlookup](#), [outputlookup](#)

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the lookup command.

makecontinuous

Synopsis

Makes a field that is supposed to be the x-axis continuous (invoked by chart/timechart).

Syntax

makecontinuous [<field>] <bucketing-option>*

Required arguments

<bucketing-option>

Datatype: bins | span | start-end

Description: Discretization options. See "Bucketing options" for details.

Optional arguments

<field>

Datatype: <field>

Description: Specify a field name.

Bucketing 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 bucket, 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 buckets. Data outside of the [start, end] range is discarded.

Log span syntax

<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).

Span length syntax

span-length

Syntax: [<timescale>]

Description: A span length based on time.

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.

<sec>

Syntax: s | sec | secs | second | seconds

Description: Time scale in seconds.

<min>

Syntax: m | min | mins | minute | minutes

Description: Time scale in minutes.

<hr>

Syntax: h | hr | hrs | hour | hours
Description: Time scale in hours.

<day>

Syntax: d | day | days
Description: Time scale in days.

<month>

Syntax: mon | month | months
Description: Time scale in months.

<subseconds>

Syntax: us | ms | cs | ds
Description: Time scale in microseconds (us), milliseconds (ms), centiseconds (cs), or deciseconds (ds).

Description

Examples

Example 1: Make "_time" continuous with a span of 10 minutes.

```
... | 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

Synopsis

Changes a specified field into a multi-value field during a search.

Syntax

`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: Defines one or more characters that separate each field value. Defaults to a single space (" ").

`tokenizer`

Syntax: `tokenizer=<string>`

Description: Defines a regex tokenizer to delimit the field values.

`allowempty`

Syntax: `allowempty=<bool>`

Description: Specifies whether or not consecutive delimiters should be treated as one. Defaults to false.

`setsv`

Syntax: `setsv=<bool>`

Description: The setsv boolean option controls if the original value of the field should be kept for the single valued version. Defaults to false.

Description

Treat specified field as multi-valued, using either a simple string delimiter (can be multicharacter), or a regex tokenizer.

Examples

Example 1: For sendmail search results, separate the values of "senders" into multiple values. Then, display the top values.

```
eventtype="sendmail" | makemv delim="," senders | top senders
```


Example 2: Separate the value of "foo" into multiple values.

```
... | makemv delim=":" allowempty=t foo
```

See also

[mvcombine](#), [mvexpand](#), [nomv](#),

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the makemv command.

map

Synopsis

Looping operator, performs a search over each search result.

Syntax

map (<searchoption>|<savedsplunkoption>) [maxsearches=int]

Required arguments

<savedsplunkoption>

Syntax: <string>

Description: Name of a saved search. No default.

<searchoption>

Syntax: [<subsearch>] | search="<string>"

Description: The search to map. The search argument can either be a subsearch to run or just the name of a saved search. The argument also supports the metavariable: `$_serial_id$`, a 1-based serial number within map of the search being executed, for example: `[search starttimeu:::$start$ endtimeu:::end source="$source$"]`. No default.

Optional arguments

maxsearches

Syntax: maxsearches=<int>

Description: The maximum number of searches to run. This will generate a message if there are more search results. Defaults to 10.

Description

For each input (each result of a previous search), the map command iterates through the field-values from that result and substitutes their value for the \$variable\$ in the search argument. For more information,

- Read "About subsearches" in the Search Manual.
- Read "How to use the search command" in the Search Manual.

Examples

Example 1: Invoke the map command with a saved search.

```
error | localize | map mytimebased_savedsearch
```

Example 2: Maps the start and end time values.

```
... | map search="search starttime::$start$ endtime::$end$"
maxsearches=10
```

Example 3: 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
```

Which returns a table of results, such as:

User	Host	Count
userA	serverA	1
userB	serverA	3
userA	serverB	2

When you 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 user's logon event. The results are returned as a table, such as:

_time	computername	computertime	username	usertime
-------	--------------	--------------	----------	----------

10/12/12 8:31:35.00 AM	Workstation\$	10/12/2012 08:25:42	userA	10/12/2012 08:31:35 AM
---------------------------	---------------	------------------------	-------	---------------------------

(Thanks to 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.

metadata

Synopsis

Returns a list of source, sourcetypes, or hosts from a specified index or distributed search peer.

Syntax

```
| metadata [type=<metadata-type>] [<index-specifier>] [<server-specifier>]
```

Optional arguments

type

Syntax: type= hosts | sources | sourcetypes

Description: Specify the type of metadata to return.

index-specifier

Syntax: index=<index_name>

Description: Specify the index from which to return results.

server-specifier

Syntax: splunk_server=<string>

Description: Specify the distributed search peer from which to return results. If used, you can specify only one `splunk_server`.

Description

The `metadata` command returns data about a specified index or distributed search peer. It returns information such as a list of the hosts, sources, or source types accumulated over time and when the first, last, and most recent event was seen for each value of the specified metadata type. It does not provide a snapshot of an index over a specific timeframe (such as last 7 days). For example, if you search for:

```
| metadata type=hosts
```

Your results will look something like this:

	firstTime ↕	host ↕	lastTime ↕	recentTime ↕	totalCount ↕	type ↕
1	1293005265	apache3.splunk.com	1293609574	1293609574	27888	hosts
2	1293005220	apache2.splunk.com	1293609574	1293609574	27705	hosts
3	1293005265	apache1.splunk.com	1293609555	1293609555	9199	hosts
4	1293055241	mysql.splunk.com	1293492848	1293492848	180	hosts

Where:

- `firstTime` is the timestamp for the first time that the indexer saw an event from this host.
- `lastTime` is the timestamp for the last time that the indexer saw an event from this host.
- `recentTime` is the `indextime` for the most recent time that the index saw an event from this host (that is, the time of the last update).
- `totalcount` is the total number of events seen from this host.
- `type` 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, `lastTime` and `recentTime` are equal. However, if the data is historical, then the values of these fields could be different.

Examples

Example 1: Return the values of "host" for events in the "_internal" index.

```
| metadata type=hosts index=_internal
```

Example 2: Return values of "sourcetype" for events in the "_audit" index on server foo.

```
| metadata type=sourcetypes index=_audit splunk_server=foo
```

See also

[dbinspect](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the metadata command.

metasearch

Synopsis

Retrieves event metadata from indexes based on terms in the <logical-expression>.

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.

<lit-value>

Syntax: <string> | <num>

Description: An exact, or literal, value of a field; used in a comparison expression.

<value>

Syntax: <lit-value> | <field>

Description: In comparison-expressions, the literal (number or string) value of a field or another field name.

Index expression

<search-modifier>

Syntax: <field-specifier>|<savedsplunk-specifier>|<tag-specifier>

Time options

Splunk allows many flexible options for searching based on time. For a list of time modifiers, see the topic ["Time modifiers for search"](#)

<timeformat>

Syntax: timeformat=<string>

Description: Set the time format for starttime and endtime terms. By default, the 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. Read more about time modifier syntax in ["Specify time modifiers in your search"](#).

Description

Retrieves event metadata from indexes based on terms in the <logical-expression>. Metadata fields include source, sourcetype, host, _time, index, and splunk_server.

Examples

Example 1: Return metadata for events with "404" and from host "webserver1".

```
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.

multikv

Synopsis

Extracts field-values from table-formatted events.

Syntax

```
multikv [conf=<stanza_name>] [<multikv-option>]*
```

Required arguments

<multikv-option>

Syntax: copyattrs=<bool> | fields <field-list> | filter <field-list> |
forceheader=<int> | multitable=<bool> | noheader=<bool> | rmorig=<bool>

Description: Options for extracting fields from tabular events.

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.

Multikv options

copyattrs

Syntax: copyattrs=<bool>

Description: Controls the copying of non-metadata attributes from the original event to extract events. Default is true.

fields

Syntax: fields <field-list>

Description: Filters out from the extracted events fields that are not in the given field list.

filter

Syntax: filter <field-list>

Description: If specified, a table row must contain one of the terms in the list before it is extracted into an event.

forceheader

Syntax: forceheader=<int>

Description: Forces the use of the given line number (1 based) as the table's header. By default a header line is searched for.

multitable

Syntax: multitable=<bool>

Descriptions: Controls whether or not there can be multiple tables in a single `_raw` in the original events. (default = true)

noheader

Syntax: noheader=<bool>

Description: Allow tables with no header. If no header fields would be named column1, column2, ... (default = false)

rmorig

Syntax: rmorig=<bool>

Description: Controls the removal of original events from the result set. (default=true)

Description

Extracts fields from events with information in a tabular format (e.g. top, netstat, ps, ... etc). A new event will be created for each table row. Field names will be derived from the title row of the table.

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

Synopsis

Run multiple searches at the same time.

Syntax

```
... | multisearch <subsearch1> <subsearch2> <subsearch3> ...
```

Required arguments

<subsearch>

Syntax:

Description: At least two streaming searches.

Description

Executes multiple **streaming** searches at the same time. Must specify at least 2 subsearches and only purely streaming operations are allowed in each subsearch (e.g. search, eval, where, fields, rex, ...)

Examples

Example 1: Search for both events from index a and b and add different fields using eval in each case.

```
... | 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

Synopsis

Combines events in the search results that have a single differing field value into one result with a multi-value field of the differing field.

Syntax

```
mvcombine [delim=<string>] <field>
```

Required arguments

field

Syntax: <field>

Description: The name of a multivalued field.

Optional arguments

delim

Syntax: delim=<string>

Description: Defines the string character to delimit each value. Defaults to a single space, (" ").

Description

For each group of results that are identical except for the given field, combine them into a single result where the given field is a multivalued field. `delim` controls how values are combined, defaulting to a space character (" ").

Examples

Example 1: Combine the values of "foo" with ":" delimiter.

```
... | mvcombine delim=":" foo
```

Example 2: Suppose you have three events that are the same except for the IP address value:

```
Nov 28 11:43:48 2010 host=datagen-host1 type=dhclient: bound to  
ip=209.202.23.154  
message= ASCII renewal in 5807 seconds.
```

```
Nov 28 11:43:49 2010 host=datagen-host1 type=dhclient: bound to  
ip=160.149.39.105  
message= ASCII renewal in 5807 seconds.
```

```
Nov 28 11:43:49 2010 host=datagen-host1 type=dhclient: bound to  
ip=199.223.167.243  
message= ASCII renewal in 5807 seconds.
```

This search returns the three IP address in one field and delimits the values with a comma, so that `ip="209.202.23.154, 160.149.39.105, 199.223.167.243"`.

```
... | mvcombine delim="," ip
```

Example 3: In a multivalued events:

```
sourcetype="WMI:WinEventLog:Security" | fields EventCode,  
Category,RecordNumber | mvcombine delim="," RecordNumber | nomv  
RecordNumber
```

See also

[makemv](#), [mvexpand](#), [nomv](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the mvcombine command.

mvexpand

Synopsis

Expands the values of a multi-value field into separate events for each value of the multi-value 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 is 0, or no limit.

Description

For each result with the specified field, create a new result for each value of that field in that result if it a multivalue field.

Examples

Example 1: Create new events for each value of multi-value field, "foo".

```
... | mvexpand foo
```

Example 2: Create new events for the first 100 values of multi-value field, "foo".

```
... | mvexpand foo limit=100
```

Example 3: The mvexpand command only works on one multivalued field. This example walks through how to expand an event with more than one multivalued field into individual events for each field's value. For example, given these events, with sourcetype=data:

```
2012-10-01 00:11:23 a=22 b=21 a=23 b=32 a=51 b=24
2012-10-01 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.

```
sourcetype=data | 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
```

	<u>time</u> ↕	fields ↕
1	10/1/12 12:11:23.000 AM	22,21 23,32 51,24
2	10/1/12 12:11:22.000 AM	1,2 2,3 5,2

Use mvexpand and the [rex command](#) on the new field, fields, to create new events and extract the fields alpha and beta:

```
sourcetype=data | 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
_time alpha beta
```

Use the [table command](#) to display only the _time, alpha, and beta fields in a

	<u>time</u> ↕	alpha ↕	beta ↕
1	10/1/12 12:11:23.000 AM	22	21
2	10/1/12 12:11:23.000 AM	23	32
3	10/1/12 12:11:23.000 AM	51	24
4	10/1/12 12:11:22.000 AM	1	2
5	10/1/12 12:11:22.000 AM	2	3
6	10/1/12 12:11:22.000 AM	5	2

results table:

(Thanks to Duncan for this example. You can see another version of this with JSON data and the [spath command](#).)

See also

[makemv](#), [mvcombine](#), [nomv](#)

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the mvexpand command.

nomv

Synopsis

Changes a specified multi-value field into a single-value field at search time.

Syntax

nomv <field>

Required arguments

field

Syntax: <field>

Description: The name of a multivalue field.

Description

Converts values of the specified multi-valued field into one single value (overrides multi-value field configurations set in fields.conf).

Examples

Example 1: For sendmail events, combine the values of the senders field into a single value; then, display the top 10 values.

```
eventtype="sendmail" | nomv senders | top senders
```

See also

[makemv](#), [mvcombine](#), [mvexpand](#), [convert](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `nomv` command.

outlier

Synopsis

Removes outlying numerical values.

Syntax

`outlier <outlier-option>* [<field-list>]`

Required arguments

`<outlier-option>`

Syntax: `<action> | <param> | <type> | <uselower>`

Description: Outlier options.

Optional arguments

`<field-list>`

Syntax: `<field>, ...`

Description: Comma-delimited list of field names.

Outlier options

`<type>`

Syntax: `type=iqr`

Description: Type of outlier detection. Currently, the only option available is IQR (inter-quartile range).

`<action>`

Syntax: `action=rm | remove | tf | transform`

Description: Specify what to do with outliers. RM | REMOVE removes the event containing the outlying numerical value. TF | TRANSFORM truncates the outlying value to the threshold for outliers and prefixes the value with "000". Defaults to tf.

<param>

Syntax: param=<num>

Description: Parameter controlling the threshold of outlier detection. For type=IQR, an outlier is defined as a numerical value that is outside of param multiplied the inter-quartile range. Defaults to 2.5.

<uselower>

Syntax: uselower=<bool>

Description: Controls whether to look for outliers for values below the median. Defaults to false|f.

Description

Removes or truncates outlying numerical values in selected fields. If no fields are specified, then outlier will attempt to process all fields.

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](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the outlier command.

outputcsv

Synopsis

Outputs search results to the specified `csv` file.

Syntax

```
outputcsv [append=<bool>] [create_empty=<bool>] [dispatch=<bool>]  
[usexml=<bool>] [singlefile=<bool>] [<filename>]
```

Optional arguments

append

Syntax: append=<bool>

Description: If 'append' is true, we will attempt to append to an existing csv file if it exists or create a file if necessary. If there is an existing file that has a csv header already, we will only emit the fields that are referenced by that header. .gz files cannot be append to. Defaults to false.

create_empty

Syntax: create_empty=<bool>

Description: If set to true and there are no results, creates a 0-length file. When false, no file is created and the files is deleted if it previously existed. Defaults to 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`. If no filename specified, rewrites the contents of each result as a CSV row into the `"_xml"` field. Otherwise writes into a file (appends ".csv" to filename if filename has no existing extension).

singlefile

Syntax: singlefile=<bool>

Description: If singlefile is set to true and output spans multiple files, collapses it into a single file.

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 specified when invoking outputcsv from the UI.

Examples

Example 1: Output search results to the CSV file 'mysearch.csv'.

```
... | outputcsv mysearch
```

See also

[inputcsv](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the outputcsv command.

outputlookup

Synopsis

Writes search results to the specified static lookup table.

Syntax

```
outputlookup [append=<bool>] [create_empty=<bool>] [max=<int>]  
[createinapp=<bool>] (<filename> | <tablename>)
```

Required arguments

<filename>

Syntax: <string>

Description: The name of the lookup 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.

Optional arguments

append

Syntax: append=<bool>

Description: If 'append' is true, we will attempt to append to an existing csv file if it exists or create a file if necessary. If there is an existing file that has a csv header already, we will only emit the fields that are referenced by that header. .gz files cannot be append to. Defaults to false.

max

Syntax: max=<int>

Description: The number of rows to output.

create_empty

Syntax: create_empty=<bool>

Description: If set to true and there are no results, creates a 0-length file. When false, no file is created and the file is deleted if it previously existed. Defaults to true.

createinapp

Syntax: createinapp=<bool>

Description: If set to false or if there is no current application context, then create the file in the system lookups directory.

Description

Saves results to a lookup table as specified by a filename (must end with .csv or .gz) or a table name (as specified by a stanza name in transforms.conf). If the lookup file does not exist, Splunk creates the file in the lookups directory of the current application. **If the lookup file already exists, Splunk overwrites that file with the results of outputlookup.** If the 'createinapp' option is set to false or if there is no current application context, then Splunk creates the file in the system lookups directory.

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
```

See also

[inputlookup](#), [lookup](#), [outputcsv](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the outputlookup command.

outputtext

Synopsis

Outputs the raw text (`_raw`) of results into the `_xml` field.

Syntax

```
outputtext [usexml=<bool>]
```

Optional arguments

usexml

Syntax: usexml=<bool>

Description: If usexml is set to true (the default), the `_raw` field is `xml` escaped.

Description

Rewrites the `_raw` field of the result into the `_xml` field. If usexml is set to true (the default), the `_raw` field is `xml` escaped.

Examples

Example 1: Output the `"_raw"` field of your current search into `"_xml"`.

... | 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/backfill summary indexes. There is script, called `fill_summary_index.py`, that will backfill your indexes or fill summary index gaps. For more information, refer to this [Knowledge Manager manual topic](#).

Synopsis

Finds events in a summary index that overlap in time or have missed events.

Syntax

`overlap`

Description

Find events in a summary index that overlap in time, or find gaps in time during which a scheduled saved search may 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 (in `etc/searchscripts/sumindexoverlap.py`), which 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 will not work. Instead, the index should contain events such as chart, stats, and timechart results.

Examples

Example 1: Find overlapping events in "summary".

```
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.

predict

Synopsis

Predict future values of fields.

Syntax

```
predict <variable_to_predict> [AS <newfield_name>] [<predict_option>]
```

Required arguments

<variable_to_predict>

Syntax: <field>

Description: The field name for the variable that you want to predict.

Optional arguments

<newfield>

Syntax: <string>

Description: Renames the field name for <variable_to_predict>.

<predict_option>

Syntax: algorithm=<algorithm_name> | correlate_field=<field> |
future_timespan=<number> | holdback=<number> | period=<number> |
lowerXX=<field> | upperYY=<field>

Description: Forecasting options. All options can be specified anywhere in any order.

Predict options

algorithm

Syntax: algorithm= LL | LLP | LLT | LLB

Description: Specify the name of the forecasting algorithm to apply: LL (local level), LLP (seasonal local level), LLT (local level trend), or LLB (bivariate local level). Each algorithm expects a minimum number of data points; for more information, see "Algorithm options" below.

correlate

Syntax: correlate=<field>

Description: For bivariate model, indicates the field to correlate against.

future_timespan

Syntax: future_timespan=<number>

Description: The length of prediction into the future. Must be a non-negative number.

holdback

Syntax: holdback=<number>

Description: Specifies not to use the last <number> of data points to build the model. Typically, this is used to compare the predicted values to the actual data.

lowerXX

Syntax: lower<int>=<field>

Description: Specifies a field name for the lower <int> percentage confidence interval. <int> is greater than or equal to 0 and less than 100. Defaults to lower95.

period

Syntax: period=<number>

Description: If algorithm=LLP, specify the seasonal period of the time series data. If not specified, the period is automatically computed. If algorithm is not LLP, this is ignored.

upperYY

Syntax: upper<int>=<field>

Description: Specifies a field name for the upper <int> percentage confidence interval. <int> is greater than or equal to 0 and less than 100. Defaults to upper95.

Algorithm options

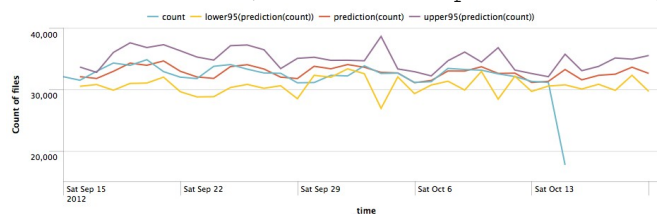
The algorithm names are: LL, LLP, LLT and LLB. The first three deal with univariate time series while the fourth deals with bivariate time series. Each algorithm above expects a minimum number of data points. If not enough effective data points are supplied, an error message will be displayed. For instance, the field itself may have more than enough data points, but the number of effective data points may be small if the holdback is large.

Algorithm option	Algorithm name	Description
LL	Local level	This is a univariate model with no trends and no seasonality. Expects a minimum of 2 data points.
LLP	Seasonal local level	This is a univariate model with seasonality. The periodicity of the time series is automatically computed. Expects a minimum twice the period in data points.
LLT	Local level trend	This is a univariate model with trend but no seasonality. Expects a minimum of 3 data points.
LLB	Bivariate local level	This is a bivariate model with no trends and no seasonality. Expects a minimum of 2 data points.

Examples

Example 1: Predict future downloads based on the previous download numbers.

```
index=download | timechart span=1d count(file) as count | predict count
```



Example 2: Predict the values of foo using LL or LLP, depending on whether foo is periodic.

```
... | timechart span="1m" count AS foo | predict foo
```


Example 3: Upper and lower confidence intervals need not be equaled.

```
... | timechart span="1m" count AS foo | predict foo as fubar  
algorithm=LL upper90=high lower97=low future_timespan=10 holdback=20
```

Example 4: Illustrates the LLB algorithm. The foo2 field is predicted by correlating it with the foo1 field.

```
... | timechart span="1m" count(x) AS foo1 count(y) AS foo2 | predict  
foo2 as fubar algorithm=LLB correlate=foo1 holdback=100
```

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

The rangemap command lets you classify ranges of values for numerical fields with more descriptive names.

Synopsis

Sets `range` field to the name of the ranges that match.

Syntax

```
rangemap field=<string> (<attribute_name>=<integer_range>)...  
[default=<string>]
```

Required arguments

`attribute_name`

Syntax: `<string>`

Description: The name or attribute for the specified numerical range.

`field`

Syntax: `field=<string>`

Description: The name of the input field. This field should be numeric.

<integer_range>

Syntax: <start>--<end>

Description: Define the starting integer and ending integer values for the range attributed to the "attribute_name" parameter. This can include negative values. For example: Dislike=-5--1, DontCare=0-0, Like=1-5.

Optional arguments

default

Syntax: default=<string>

Description: If the input field doesn't match a range, use this to define a default value. If you don't define a value, defaults to "None".

Description

Sets 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, all the values that apply are shown in the events. For example, if `low=1-10`, `elevated=5-15`, and the input field value is 10, then `range=low elevated`.

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
```

Using rangemap with single value panels

The Single Value dashboard panel type can be configured to use `rangemap` values; for example, Splunk ships with CSS that defines colors for low, elevated, and severe. You can customize the CSS for these values to apply different colors. Also, you have to edit the XML for the view to associate the colors with the `range` value; to do this:

1. Go to **Manager >> User interface >> Views** and select the view you want to edit.
2. For the single value panel that uses the rangemap search, include the following line underneath the `<title />` tags:

```
<option name="classField">range</option>
```

So, if you had a view called "Example" and your search was named, "Count of events", your XML might look something like this:

```
<?xml version='1.0' encoding='utf-8'?>
<dashboard>
  <label>Example</label>
  <row>
    <single>
      <searchName>Count of events</searchName>
      <title>Count of events</title>
      <option name="classField">range</option>
    </single>
  </row>
</dashboard>
```

See also

[eval](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the rangemap command.

rare

Synopsis

Displays the least common values of a field.

Syntax

rare <top-opt>* <field-list> [<by-clause>]

Required arguments

<field-list>

Syntax: <string>,...

Description: Comma-delimited list of field names.

<top-opt>

Syntax: countfield=<string> | limit=<int> | percentfield=<string> |
showcount=<bool> | showperc=<bool>

Description: Options for rare (same as top).

Optional arguments

<by-clause>

Syntax: by <field-list>

Description: The name of one or more fields to group by.

Top options

countfield

Syntax: countfield=<string>

Description: Name of a new field to write the value of count, default is "count".

limit

Syntax: limit=<bool>

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 is "percent".

showcount

Syntax: showcount=<bool>

Description: Specify whether to create a field called "count" (see "countfield" option) with the count of that tuple. Default is 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 is true.

Description

Finds the least frequent tuple of values of all fields in the field list. If optional by-clause is specified, this command will return rare tuples of values for each distinct tuple of values of the group-by fields.

Examples

Example 1: Return the least common values of the "url" field.

```
... | rare url
```

Example 2: Find the least common "user" value for a "host".

```
... | 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.

regex

Synopsis

Removes or keeps results that match the specified regular expression.

Syntax

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

Required arguments

<regex-expression>

Syntax: "<string>"

Description: A Perl Compatible Regular Expression supported by the PCRE library. Quotes are required.

Optional arguments

<field>

Syntax: <field>

Description: Specify the field name from which to match the values against the regular expression. If no field is specified, the match is against "_raw".

Description

The regex command removes results that do not match the specified regular expression. You can specify for the regex to keep results that match the expression (*field=regex-expression*) or to keep those that do not match (*field!=regex-expression*).

Note: If you want to use the "OR" ("|") command in a regex argument, the whole regex expression must be surrounded by quotes (that is, regex "expression").

Examples

Example 1: Keep only search results whose "_raw" field contains IP addresses in the non-routable class A (10.0.0.0/8).

```
... | regex _raw="(?!\\d)10\\.\\d{1,3}\\\\.\\d{1,3}\\\\.\\d{1,3}(?!\\d) "
```

Example 3: Example usage

```
... | regex _raw="complicated|regex(?=expression) "
```

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

Synopsis

Calculates how well the event matches the query.

Syntax

relevancy

Description

Calculates the 'relevancy' field based on how well the events `_raw` field matches the keywords of the 'search'. 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.

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

Synopsis

Creates a relative time field, called 'reltime', and sets it to a human readable value of the difference between 'now' and '_time'.

Syntax

reltime

Description

Sets the 'reltime' 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", etc.

Examples

Example 1: Add a reltime field.

```
... | 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

Use the `rename` command to rename fields. This command is useful for giving fields more meaningful names, such as "Product ID" instead of "pid". If you want to rename multiple fields, you can use wildcards.

Synopsis

Renames a specified field or multiple fields.

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. Can be wildcarded.

Description

Use quotes to rename a field to a phrase:

```
... | rename SESSIONID AS sessionId
```

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 Example 2, below.

Note: You cannot rename one field with multiple names. For example if you had a field A, you can't do "A as B, A as C" in one string.

```
... | stats first(host) AS site, first(host) AS report
```

Note: You do not want to use this command to merge multiple fields into one field. For example, if you had 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. Instead, see [the eval command](#) and [coalesce\(\) function](#).

Examples

Example 1: Rename the "`_ip`" field as "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.

```
... | rename count as "CountofEvents"
```

See also

[fields](#), [table](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the rename command.

replace

Synopsis

Replaces values of specified fields with a specified new value.

Syntax

```
replace (<wc-str> with <wc-str>)+ [in <field-list>]
```

Required arguments

wc-string

Syntax: <string>

Description: Specify one or more field values and their replacements. You can include wildcards to match.

Optional arguments

field-list

Syntax: <string>

Description: Specify a comma-delimited list of field names in which to do the field value replacement.

Description

Replaces a single occurrence of the first string with the second within the specified fields (or all fields if none were specified). Non-wildcard replacements specified later take precedence over those specified earlier. For wildcard replacement, fuller matches take precedence over lesser matches. To assure precedence relationships, one is advised to split the replace into two separate

invocations. When using wildcarded 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

Example 1: Change any host value that ends with "localhost" to "localhost".

```
... | replace *localhost with localhost in host
```

Example 2: Example usage.

```
... | replace "* localhost" with "localhost *" in host
```

Example 3: Change the value of two fields.

```
... | replace aug with August in start_month end_month
```

Example 5: Replace an IP address with a more descriptive name.

```
... | replace 127.0.0.1 with localhost in host
```

Example 6: Replace values of a field with more descriptive names.

```
... | replace 0 with Critical, 1 with Error in msg_level
```

Example 7: Search for an error message and replace empty strings with a whitespace. **Note:** This example won't 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
```

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

Synopsis

Access a REST endpoint and display the returned entities as search results.

Syntax

```
rest <rest-uri> [<splunk-server>=<string>] [timeout=<int>]  
(<get-arg-name>=<get-arg-value>)...
```

Required arguments

rest-uri

Syntax: <uri>

Description: URI path to the REST endpoint.

get-arg-name

Syntax: <string>

Description: REST argument name.

get-arg-value

Syntax: <string>

Description: REST argument value.

Optional arguments

splunk-server

Syntax: splunk_server=<string>

Description: Optional, argument specifies whether or not to limit results to one specific server. Use "local" to refer to the search head.

timeout

Syntax: timeout=<int>

Description: Specify the timeout in seconds when waiting for the REST endpoint to respond. Defaults to 60 seconds.

Examples

Example 1: Access saved search jobs.

```
| rest /services/search/jobs count=0 splunk_server=local | search  
isSaved=1
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has about using the `return` command.

return

Synopsis

Returns values from a subsearch.

Syntax

```
return [<count>] [<alias>=<field>] [<field>] [$<field>]
```

Arguments

<count>

Syntax: <int>

Description: Specify the number of rows. Defaults to 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.

<field>

Syntax: <field>

Description: Specify the field to return.

<\$field>

Syntax: <\$field>

Description: Specify the field values to return.

Description

The `return` command is for passing values up from a subsearch. 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 `head` and the resulting fields with the `fields`.

The command also allows convenient outputting of `field=value`, `'return source'`, `alias=value`, `'return ip=srcip'`, and `value`, `'return $srcip'`.

The `return` command defaults to using as input just the first row of results that are passed to it. Multiple rows can be specified with `count`, for example `'return 2 ip'`; and each row is ORed, 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`.

Examples

Example 1: Search for `'error ip=<someip>'`, where `someip` is the most recent ip used by Boss.

```
error [ search user=boss | return ip ]
```

Example 2: Search for `'error (user=user1 ip=ip1) OR (user=user2 ip=ip2)'`, where users and IPs come from the two most-recent logins.

```
error [ search login | return 2 user, ip ]
```

Example 3: Return to eval the `userid` of the last user, and increment it by 1.

```
... | 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

Synopsis

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.

Syntax

reverse

Examples

Example 1: Reverse the order of a result set.

```
... | 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

Synopsis

Specifies a Perl regular expression named groups to extract fields while you search.

Syntax

```
rex [field=<field>] (<regex-expression> [max_match=<int>] | mode=sed  
<sed-expression>)
```

Required arguments

field

Syntax: field=<field>

Description: The field that you want to extract information from.

regex-expression

Syntax: "<string>"

Description: A Perl Compatible Regular Expression supported by the PCRE library. Quotes are required.

sed-expression

Syntax: "<string>"

Description: Use Unix sed syntax to replace strings or substitute characters. For more information, see Anonymize data in the *Getting Data In manual*. Quotes are required.

Optional arguments

max_match

Syntax: max_match=<int>

Description: Controls the number of times the regex is matched. If greater than 1, the resulting fields will be multivalued fields. Defaults to 1, use 0 to mean unlimited.

Description

Matches the value of the field against the unanchored regex and extracts the Perl regex named groups into fields of the corresponding names. If mode is set to 'sed' the given sed expression will be applied to the value of the chosen field (or to `_raw` if a field is not specified).

Examples

Example 1: Extract "from" and "to" fields using regular expressions. If a raw event contains "From: Susan To: Bob", then `from=Susan` and `to=Bob`.

```
... | rex field=_raw "From: (?<from>.*) To: (?<to>.*)"
```

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
```

```
... | 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.

```
... | rex mode=sed "s/(\d{4}-){3}/XXXX-XXXX-XXXX-/g"
```


See also

[extract](#), [kvform](#), [multikv](#), [regex](#), [spath](#), [xmlkv](#),

rtorder

Synopsis

Buffers events from real-time search to emit them in ascending time order when possible.

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 is 10 seconds.

discard

Syntax: discard=<bool>

Description: Specifies whether or not to always discard out-of-order events. Default is false.

max_buffer_size

Syntax: max_buffer_size=<int>

Description: Specifies the maximum size of the buffer. Default is 50000, or the `max_result_rows` setting of the [search] stanza in limits.conf.

Description

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 only after the current time reaches at least the span of time given by `buffer_span` after the timestamp of the event.

Events will also be 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, that out of order event will be emitted immediately unless the discard option is set to true. When discard is set to true, out of order events will always be discarded, assuring that the output is always strictly in time ascending order.

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.

```
... | 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

See [script](#).

savedsearch

Synopsis

Returns the search results of a saved search.

Syntax

`savedsearch <savedsearch name> [<savedsearch-opt>]*`

Required arguments

savedsearch name

Syntax: <string>

Description: Name of the saved search to run.

savedsearch-opt

Syntax: <macro>|<replacementt>

Description: The savedsearch options lets you specify either no substitution or the key/value pair to use in the macro replacement.

Savedsearch options

macro

Syntax: nosubstitution=<bool>

Description: If true, no macro replacements are made. Defaults to false.

replacement

Syntax: <field>=<string>

Description: A key/value pair to use in macro replacement.

Description

Runs a saved search, possibly cached by disk. Also, performs macro replacement.

Examples

Example 1: Run the "mysecurityquery" saved search.

```
| savedsearch mysecurityquery
```

See also

[search](#)

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the savedsearch command.

script

Synopsis

Makes calls to external Perl or Python programs.

Syntax

script (perl|python) <script-name> [<script-arg>]* [maxinputs=<int>]

Required arguments

script-name

Syntax: <string>

Description: The name of the script to execute, minus the path and file extension.

Optional arguments

maxinputs

Syntax: maxinputs=<int>

Description: Determines how many of the top results are passed to the script. Defaults to 100.

script-arg

Syntax: <string>

Description: One or more arguments to pass to the script. If passing more than one argument, delimit each with a space.

Description

Calls an external python or perl program that can modify or generate search results. Scripts must live in `splunk_home/etc/searchscripts` and only a search user with administrator privileges may execute them. If the script is a custom search command, it should be located in

`$SPLUNK_HOME/etc/apps/<app_name>/bin/`. To invoke the script:

- For python, use `splunk_home/bin/python`
- For perl, use `/usr/bin/perl`

Examples

Example 1: Run the Python script "myscript" with arguments, myarg1 and myarg2; then, email the results.

```
... | script python 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

Synopsis

Anonymizes the search results.

Syntax

```
scrub [public-terms=<filename>] [private-terms=<filename>]  
[name-terms=<filename>] [dictionary=<filename>] [timeconfig=<filename>]
```

Optional arguments

public-terms

Syntax: public-terms=<filename>

Description: Specify a filename that includes the public terms to be anonymized.

private-terms

Syntax: private-terms=<filename>

Description: Specify a filename that includes the private terms to be anonymized.

name-terms

Syntax: name-terms=<filename>

Description: Specify a filename that includes names to be anonymized.

dictionary

Syntax: dictionary=<filename>

Description: Specify a filename that includes a dictionary of terms to be anonymized. Defaults to dictionary and configuration files found in `$SPLUNK_HOME/etc/anonymizer`.

timeconfig

Syntax: timeconfig=<filename>

Description: Specify a filename that includes time configurations to be anonymized.

Description

Anonymizes the search results by replacing identifying data - usernames, ip addresses, domain names, etc. - with fictional values that maintain the same word length. For example, it may 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. By default the dictionary and configuration files found in `$splunk_home/etc/anonymizer` are used. These can be overridden by specifying arguments to the scrub command. The arguments exactly correspond to the settings in the stand-alone `splunk anonymize` command, and are documented there.

Anonymizes all attributes, exception those that start with `_` (except `_raw`) or `date_`, or the following attributes: `eventtype`, `linecount`, `punct`, `sourcetype`, `timeendpos`, `timestartpos`.

Examples

Example 1: Anonymize the current search results.

```
... | scrub
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the scrub command.

search

Use the `search` command to retrieve events from your indexes, using keywords, quoted phrases, wildcards, and key/value expressions. The command is implicit when it's the first search command (used at the beginning of a pipeline). When it's not the first command in the pipeline, it's used to filter the results of the

previous command.

After you retrieve events, you can apply commands to them to transform, filter, and report on them. Use the vertical bar "|", or pipe character, to apply a command to the retrieved events.

Synopsis

Retrieve events from indexes or filter the results of a previous search command in the pipeline.

Syntax

search <logical-expression>

Arguments

<logical-expression>

Syntax: <time-opts> | <search-modifier> | [NOT] <logical-expression> | <index-expression> | <comparison-expression> | <logical-expression> [OR] <logical-expression>

Description: Includes all keywords or key/value pairs used to describe the events to retrieve from the index. These filters can be defined using Boolean expressions, comparison operators, time modifiers, search modifiers, or combinations of 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>

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

<cmp>

Syntax: = | != | < | <= | > | >=

Description: Comparison operators. You can use comparison expressions when searching field/value pairs. Comparison expressions with "=" and "!=" work with all field/value pairs. Comparison expressions with < > <= >= work only with fields that have numeric values.

<field>

Syntax: <string>

Description: The name of a field.

<lit-value>

Syntax: <string> | <num>

Description: An exact or literal value of a field. Used in a comparison expression.

<value>

Syntax: <lit-value> | <field>

Description: In comparison-expressions, the literal (number or string) value of a field or another field name.

Index expression

<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 searches the `_raw` field for the matching events or results.

<search-modifier>

Syntax:

<sourcetype-specifier>|<host-specifier>|<source-specifier>|<savedsplunk-specifier>|<eventtype-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></code>`.

- Read more about searching with default fields in the *Knowledge Manager manual*.

- Read more about using tags and field alias in the *Knowledge Manager manual*.

Time options

Splunk allows many flexible options for searching based on time. For a list of time modifiers, see the topic ["Time modifiers for search"](#)

<timeformat>

Syntax: timeformat=<string>

Description: Set the time format for starttime and endtime terms. By default, the timestamp is formatted: `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. Read more about this time modifier syntax in "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.

Description

The search command enables you to use keywords, phrases, fields, boolean expressions, and comparison expressions to specify exactly which events you want to retrieve from a Splunk index(es).

Some examples of search terms are:

- keywords: `error login`
- quoted phrases: `"database error"`
- boolean operators: `login NOT (error OR fail)`
- wildcards: `fail*`

- field values: `status=404`, `status!=404`, or `status>200`

Read more about how to "Use the search command to retrieve events" in the Search Manual.

Quotes and escaping characters

Generally, you need quotes around phrases and field values that include white spaces, commas, pipes, quotes, and/or brackets. Quotes must be balanced, an opening quote must be followed by an unescaped closing quote. 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 quotes around keywords and phrases if you don't 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 quotes. 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 quote 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 when specifying search phrases. TERM forces Splunk to match whatever is inside the parentheses as a single term in the index, even if it contains characters that are usually recognized as breaks or delimiters (such as underscores and spaces).

If you searched for the quoted phrase "error_type", Splunk ends up searching for "error" and "type" and post filtering the results. This would also include events that contained "error_type" as segments of other keywords or phrases, for example "error_type.default" or "this_error_type". If you use TERM(error_type), you force Splunk to exclude these other keywords.

Search with CASE()

You can use the CASE() directive to search for terms and field values that are case-sensitive.

Examples

The following are just a few examples of how to use the `search` command. You can find more examples in the Start Searching topic of the *Splunk Tutorial*.

Example 1: This example demonstrates key/value pair matching for specific values of source IP (src) and destination IP (dst).

```
src="10.9.165.*" OR dst="10.9.165.8"
```

Example 2: This example demonstrates key/value pair matching with boolean and comparison operators. Search for events with code values of either 10 or 29, any host that isn't "localhost", and an `xqp` value that is greater than 5.

```
(code=10 OR code=29) host!="localhost" xqp>5
```

Example 3: This example demonstrates key/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*)
```

Example 4: This example demonstrates how to use `search` later in the pipeline to filter out search 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 | transactions clientip startswith="login"  
endswith="logout" | search eventcount>3
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the search command.

searchtxn

Synopsis

Finds transaction events within specified search constraints.

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 `transactiontype` 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. Defaults to 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 will speed up search, favoring more recent values. Defaults to 1000.

use_disjunct

Syntax: use_disjunct=<bool>

Description: Determines if each term in SEARCH-STRING should be ORed on the initial search. Defaults to true.

Description

Retrieves events matching the transaction type `transaction-name` with events transitively discovered by the initial event constraint of the `search-string`.

For example, given an 'email' transactiontype with fields="qid pid" and with a search attribute of 'sourcetype="sendmail_syslog"', and a `search-string` of "to=root", `searchtxn` will find all the events that match 'sourcetype="sendmail_syslog" to=root'.

From those results, all the qid's and pid's are transitively used to find further search for relevant events. When no more qid or pid values are found, the resulting search is run:

```
'sourcetype="sendmail_syslog" ((qid=val1 pid=val1) OR (qid=valn pid=valm) |
transaction name=email | search to=root'
```

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.

selfjoin

Synopsis

Joins results with itself.

Syntax

selfjoin [<selfjoin-options>]* <field-list>

Required arguments

<field-list>

Syntax: <field>...

Description: Specify the field or list of fields to join on.

<selfjoin-options>

Syntax: overwrite=<bool> | max=<int> | keepsingle=<bool>

Description: Options for the selfjoin command. You can use a combination of the three options.

Selfjoin options

keepsingle

Syntax: keepsingle=<bool>

Description: Controls whether or not results with a unique value for the join fields (which means, they have no other results to join with) should be retained. Defaults to false.

max

Syntax: max=<int>

Description: Indicate the maximum number of 'other' results to join with each main result. If 0, there is no limit. Defaults to 1.

overwrite

Syntax: overwrite=<bool>

Description: Specify if fields from these 'other' results should overwrite fields of the results used as the basis for the join. Defaults to true.

Description

Join results with itself, based on a specified field or list of fields to join on. The selfjoin options, `overwrite`, `max`, and `keepsingle` controls the out results of the selfjoin.

Examples

Example 1: Join results with itself on 'id' field.

```
... | selfjoin id
```

See also

[join](#)

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the selfjoin command.

set

Synopsis

Performs set operations on **subsearches**.

Syntax

```
set (union|diff|intersect) subsearch subsearch
```

Required arguments

subsearch

Syntax: <string>

Description: Specifies a subsearch. For more information about subsearch syntax, see "About subsearches" in the Search manual.

Description

Performs two subsearches and then executes the specified set operation on the two sets of search results:

- The result of a **union operation** are events that result from either subsearch.
- The result of a **diff operation** are the events that result from either subsearch that are not common to both.

- The result of an **intersect operation** are the events that are common for both subsearches.

Important: The set command works on less than 10 thousand results.

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 don't 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](#), [join](#), [diff](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the set command.

setfields

Synopsis

Sets the field values for all results to a common value.

Syntax

setfields <setfields-arg>, ...

Required arguments

<setfields-arg>

Syntax: string="<string>"

Description: A key-value pair with quoted value. Standard key cleaning will be performed, ie all non-alphanumeric characters will be replaced with '_' and leading '_' will be removed.

Description

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.

Examples

Example 1: Specify a value for the ip and foo fields.

```
... | 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.

sendemail

Synopsis

Emails search results to specified email addresses.

Syntax

sendemail to=<email_list> [from=<email_list>] [cc=<email_list>]
[bcc=<email_list>] [format= (html|raw|text|csv)] [inline= (true|false)]
[sendresults=(true|false)] [sendpdf=(true|false)] [priority=
(highest|high|normal|low|lowest)] [server=<string>]
[width_sort_columns=(true|false)] [graceful=(true|false)] [sendresults=<bool>]
[sendpdf=<bool>]

Required arguments

to

Syntax: to=<email_list>

Description: List of email addresses to send search results to.

Optional arguments

bcc

Syntax: bcc=<email_list>

Description: Blind cc line; comma-separated and quoted list of valid email addresses.

cc

Syntax: cc=<email_list>

Description: Cc line; comma-separated quoted list of valid email addresses.

format

Syntax: format= csv | html | raw |text

Description: Specifies how to format the email's contents. Defaults to HTML.

from

Syntax: from=<email_list>

Description: Email address from line. Defaults to "splunk@<hostname>".

inline

Syntax: inline= true | false

Description: Specifies whether to send the results in the message body or as an attachment. Defaults to true.

graceful

Syntax: graceful= true | false

Description: If set to true, no error is thrown, if email sending fails and thus the search pipeline continues execution as if sendemail was not there.

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; defaults to normal or 3.

sendpdf

Syntax: sendpdf=true | false

Description: Specify whether to send the results with the email as an attached PDF or not. For more information about using Splunk's integrated PDF generation functionality, see "Upgrade PDF printing for Splunk Web" in the Installation Manual.

sendresults

Syntax: sendresults=true | false

Description: Determines whether the results should be included with the email. Defaults to false.

server

Syntax: server=<string>

Description: If the SMTP server is not local, use this to specify it. Defaults to **localhost**.

subject

Syntax: subject=<string>

Description: Specifies the subject line. Defaults to "Splunk Results".

width_sort_columns

Syntax: width_sort_columns=<bool>

Description: This is only valid when `format=text`. Specifies whether the columns should be sorted by their width.

Examples

Example 1: Send search results in HTML format with the subject "myresults".

```
... | sendemail to="elvis@splunk.com,john@splunk.com" format=html  
subject=myresults server=mail.splunk.com sendresults=true
```

Example 2: Send search results to the specified email.

```
... | sendemail to="elvis@splunk.com" sendresults=true
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the sendemail command.

sichart

Synopsis

Summary indexing friendly versions of chart command.

Syntax

sichart *chart_syntax*

Arguments

Refer to the [chart command syntax](#).

Description

Summary indexing friendly versions of chart command, using the same syntax. Does not require explicitly knowing what statistics are necessary to store to the summary index in order to generate a report.

Does require the chart command used to process this data have the exact same arguments as were used with the sichart command to generate the data.

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

Synopsis

Summary indexing friendly versions of `rare` command.

Syntax

```
sirare rare_syntax
```

Arguments

Refer to the [rare command syntax](#).

Description

Summary indexing friendly versions of `rare` command, using the same syntax. Does not require explicitly knowing what statistics are necessary to store to the summary index in order to generate a report.

Does require the `rare` command used to process this data have the exact same arguments as were used with the `sirare` command to generate the data.

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

Synopsis

Summary indexing friendly versions of stats command.

Syntax

```
sistats stats_syntax
```

Arguments

Refer to the [stats command syntax](#).

Description

Summary indexing friendly versions of stats command, using the same syntax. Does not require explicitly knowing what statistics are necessary to store to the summary index in order to generate a report.

Does require the stats command used to process this data have the exact same arguments as were used with the sistats command to generate the data.

Examples

Example 1: Compute the necessary information to later do 'stats avg(foo) by bar' on summary indexed results

```
... | sistats avg(foo) by bar
```

See also

[collect](#), [overlap](#), [sichart](#), [sirare](#), [sitop](#), [sitimechart](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `sistats` command.

sitimechart

Synopsis

Summary indexing friendly versions of `timechart` command.

Syntax

`sitimechart` *timechart_syntax*

Arguments

Refer to the [timechart command syntax](#).

Description

Summary indexing friendly versions of `timechart` command, using the same syntax. Does not require explicitly knowing what statistics are necessary to store to the summary index in order to generate a report.

Does require the `timechart` command used to process this data have the exact same arguments as were used with the `sitimechart` command to generate the data.

Examples

Example 1: Compute the necessary information to later do '`timechart avg(foo) by bar`' on summary indexed results.

```
... | 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

Synopsis

Summary indexing friendly versions of top command.

Syntax

`sitop top_syntax`

Arguments

Refer to the [top command syntax](#).

Description

Summary indexing friendly versions of top command, using the same syntax. Does not require explicitly knowing what statistics are necessary to store to the summary index in order to generate a report.

Does require the top command used to process this data have the exact same arguments as were used with the sitop command to generate the data.

Examples

Example 1: Compute the necessary information to later do 'top foo bar' on summary indexed results.

```
... | sitop foo bar
```

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.

sort

Synopsis

Sorts search results by the specified fields.

Syntax

sort [<count>] (<sort-by-clause>)+ [desc]

Required arguments

<count>

Syntax: <int>

Description: Specify the number of results to sort. If no count is specified, the default limit of 10000 is used. If "0" is specified, all results will be returned.

<sort-by-clause>

Syntax: (- | +) <sort-field>

Description: List of fields to sort by and their order, descending (-) or ascending (+).

Optional arguments

desc

Syntax: d | desc

Description: A trailing string that reverses the results.

Sort field options

<sort-field>

Syntax: <field> | auto(<field>) | str(<field>) | ip(<field>) | num(<field>)

Description: Options for sort-field.

<field>

Syntax: <string>

Description: The name of field to sort.

auto

Syntax: auto(<field>)

Description: Determine automatically how to sort the field's values.

ip

Syntax: ip(<field>)

Description: Interpret the field's values as an IP address.

num

Syntax: num(<field>)

Description: Treat the field's values as numbers.

str

Syntax: str(<field>)

Description: Order the field's values lexicographically.

Description

The `sort` command sorts the results by the given list of 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 results will be returned.

By default, `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 lexicographic ordering. 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 (ascending or descending).
- Alphanumeric strings are sorted based on the data type of the first character. If it starts with a number, it's sorted numerically based on that number alone; otherwise, it's 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 may be lexicographical, while for other pairs the order may be numerical. For example, if sorting in descending order: 10.1 > 9.1, but 10.1.a < 9.1.a.

Examples

Example 1: Sort results by "ip" value in ascending order and then by "url" value in descending order.

```
... | sort ip, -url
```

Example 2: Sort first 100 results in descending order of the "size" field and then by the "source" value in ascending order.

```
... | sort 100 -size, +source
```

Example 3: Sort results by the "_time" field in ascending order and then by the "host" value in descending order.

```
... | sort _time, -host
```

Example 4: Change the format of the event's time and sort the results in descending order by new time.

```
... | bucket _time span=60m | eval Time=strftime(_time,
"%m/%d %H:%M %Z") | stats avg(time_taken) AS AverageResponseTime BY
Time | sort - Time
```

(Thanks to Ayn for this example.)

Example 5. 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:

Day	Total
Friday	120
Monday	93
Tuesday	124
Thursday	356
Weekend	1022
Wednesday	248

Sorting on the day field (Day) returns a table sorted alphabetically, which doesn't 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 Ant1D and Ziegfried for this example.)

See also

[reverse](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the sort command.

spath

The `spath` command--the "s" stands for Splunk (or structured) -- provides a straightforward means for extracting information from structured data formats, XML and JSON. It also highlights the syntax in the displayed events list.

You can also use the `eval` command's `spath()` function. For more information, see the [Functions for eval and where](#).

Synopsis

Extracts values from structured data (XML or JSON) and stores them in a field or fields.

Syntax

```
spath [input=<field>] [output=<field>] [path=<datapath> | <datapath>]
```

Optional arguments

input

Syntax: input=<field>

Description: The field to read in and extract values. Defaults to `_raw`.

output

Syntax: output=<field>

Description: If specified, the value extracted from the path is written to this field name.

path

Syntax: path=<datapath> | <datapath>

Description: The location path to the value that you want to extract. If you don't use the `path` argument, the first unlabeled argument will be used as a path. A location path is composed of one or more location steps, separated by periods; for example '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 data's position in an array (this will differ 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. If you don't specify an output argument, this path becomes the field name for the extracted value.

Description

When called with no path argument, `spath` runs in "auto-extract" mode, where it finds and extracts all the fields from the first 5000 characters in the input field (which defaults to `_raw` if another input source isn't 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 it is provided).

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 indicated by curly brackets around an integer or a string. Array indices mean different things in XML and JSON. For example, in JSON, `foo.bar{3}` refers to the third element of the `bar` child of the `foo` element. In XML, this same path refers to the third `bar` child of `foo`.

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`, etc. 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 will get you all the books in the catalog).

Examples

Example 1: GitHub

As an administrator of a number of large git repositories, I want to:

- see who has committed the most changes and to which repository
- produce a list of the commits submitted for each user

I set up Splunk to track all the post-commit JSON information, then use `spath` to extract fields that I call `repository`, `commit_author`, and `commit_id`:

```
... | spath output=repository path=repository.url
... | spath output=commit_author path=commits.author.name
... | spath output=commit_id path=commits.id
```

Now, if I want to see who has committed the most changes to a repository, I can run the search:

```
... | top commit_author by repository
```

and, to see the list of commits by each user:

```
... | stats values(commit_id) by commit_author
```

Example 2: Extract a subset of an attribute

This example shows how to extract values from XML attributes and elements.

```
<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>
    </desc>
  </product>
</vendorProductSet>
```

```

        <locDesc locale="fr">Prix</locDesc>
        <locDesc locale="de">Preise</locDesc>
        <locDesc locale="ca">Preus</locDesc>
        <locDesc locale="pt">Preços</locDesc>
    </desc>
</product>

```

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}
```

Example 3: Extract and expand JSON events with multivalued 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`:

```

{"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 and rename them in a table:

```
sourcetype=json | spath | rename widget.text.size AS size,
widget.text.data AS data | table _time,size,data
```

<u>time</u>	<u>size</u>	<u>data</u>
2012-10-18 14:45:46.000 BST	36	Click here
	37	Learn more
	38	Help

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
```

_time	data	size	x
2012-10-18 14:45:46.000 BST	Click here	36	Click here,36
	Learn more	37	Learn more,37
	Help	38	Help,38

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
```

_time	data	size
2012-10-18 14:45:46.000 BST	Click here	36
2012-10-18 14:45:46.000 BST	Learn more	37
2012-10-18 14:45:46.000 BST	Help	38

(Thanks to Genti for this example.)

More examples

Example 1:

```
... | spath output=myfield path=foo.bar
... | spath output=myfield path=foo{1}
... | spath output=myfield path=foo.bar{7}.baz
```

Example 2:

```
... | spath output=author path=book{@author}
```

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

Synopsis

Provides statistics, grouped optionally by field.

Syntax

Simple: `stats (stats-function(field) [as field])+ [by field-list]`

Complete: `stats [allnum=<bool>] [delim=<string>] (<stats-agg-term> | <sparkline-agg-term>) [<by clause>]`

Required arguments

stats-agg-term

Syntax: `<stats-func>(<evaluated-field> | <wc-field>) [AS <wc-field>]`

Description: A statistical specifier optionally renamed to a new field name. The specifier can be by an aggregation function applied to a field or set of fields or an aggregation function applied to an arbitrary eval expression.

sparkline-agg-term

Syntax: `<sparkline-agg> [AS <wc-field>]`

Description: A sparkline specifier optionally renamed to a new field.

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 is false.)

delim

Syntax: `delim=<string>`

Description: Used to specify how the values in the list() or values() aggregation are delimited. (default is a single space.)

by clause

Syntax: by <field-list>

Description: The name of one or more fields to group by.

Stats function options

stats-function

Syntax: avg() | c() | count() | dc() | distinct_count() | earliest() | estdc() | estdc_error() | exactperc<int>() | first() | last() | latest() | list() | max() | median() | min() | mode() | p<in>() | perc<int>() | range() | stdev() | stdevp() | sum() | sumsq() | upperperc<int>() | values() | var() | varp()

Description: Functions used with the stats command. Each time you invoke the `stats` command, you can use more than one function; however, you can only use one `by clause`. For a list of stats functions with descriptions and examples, see ["Functions for stats, chart, and timechart"](#).

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.

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 bucket.

Description

Calculate aggregate statistics over the dataset, similar to SQL aggregation. If called without a `by` clause, one row is produced, which represents the aggregation over the entire incoming result set. If called with a `by`-clause, one row is produced for each distinct value of the `by`-clause.

Examples

Example 1

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 **Get the sample data into Splunk** and follow the instructions. Then, run this search using the time range, **Other > Yesterday**.

Count the number of different types of requests made against 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 field values for the `stats` command to count. The search is only interested in two page request methods, GET or POST. The first clause tells Splunk to count the Web access events that contain the `method=GET` field value and call the result "GET". The second clause does the same for `method=POST` events. Then the `by` clause, `by host`, separates the counts for each request by the `host` value that they correspond to.

This returns the following table:

	host ↕	GET ↕	POST ↕
1	apache1.splunk.com	1152	169
2	apache2.splunk.com	3771	154
3	apache3.splunk.com	3855	176

Note: You can use the `stats`, `chart`, and `timechart` commands to perform the same statistical calculations on your data. The `stats` command returns a table of results. The `chart` command returns the same table of results, but you can use the Report Builder to format this table as a chart. If you want to chart your results over a time range, use the `timechart` command. You can also see variations of this example with the [chart](#) and [timechart](#) commands.

Example 2

This example uses recent (September 23-29, 2010) earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains the source network (Src), ID (Eqid), version, date, location, magnitude, depth (km) and number of reporting stations (NST) for each earthquake over the last 7 days.

Download the text file, **M 1+ earthquakes, past 7 days**, save it as a CSV file, and upload it to Splunk. Splunk should extract the fields automatically.

Search for earthquakes in and around California and count the number of quakes that were recorded. Then, calculate the minimum, maximum, the range (difference between the min and max), and average magnitudes of those recent quakes.

```
source=eqs7day-M1.csv Region=*California | stats count, max(Magnitude), min(Magnitude), range(Magnitude), avg(Magnitude) by Region
```

Use `stats` functions for each of these calculations: `count()`, `max()`, `min()`, `range()`, and `avg()`. This returns the following table:

10 results over all time

Options... Results per page 10

Overlay: None

	Region	count	max(Magnitude)	min(Magnitude)	range(Magnitude)	avg(Magnitude)
1	Central California	80	2.7	1.0	1.7	1.368750
2	Greater Los Angeles area, California	12	2.5	1.0	1.5	1.450000
3	Gulf of California	1	4.2	4.2	0.0	4.200000
4	Gulf of Santa Catalina, California	1	1.9	1.9	0.0	1.900000
5	Long Valley area, California	8	1.7	1.0	0.7	1.237500
6	Northern California	102	2.8	1.0	1.8	1.506863
7	San Francisco Bay area, California	20	2.1	1.0	1.1	1.425000
8	Southern California	120	2.9	1.0	1.9	1.415000
9	offshore Central California	2	2.2	1.5	0.7	1.850000
10	offshore Northern California	6	2.6	1.1	1.5	1.800000

There were 870 events for this data set. From these results, you can see that approximately 350 of those recorded earthquakes occurred in and around California--!!!

Example 3

This example uses recent (September 23-29, 2010) earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains the source network (Src), ID (Eqid), version, date, location, magnitude, depth (km) and number of reporting stations (NST) for each earthquake over the last 7 days.

Download the text file, **M 1+ earthquakes, past 7 days**, and upload it to Splunk. Splunk should extract the fields automatically.

Search for earthquakes in and around California and calculate the mean, standard deviation, and variance of the magnitudes of those recent quakes.

```
source=eqs7day-M1.csv Region=*California | stats mean(Magnitude),  
stdev(Magnitude), var(Magnitude) by Region
```

Use `stats` functions for each of these calculations: `mean()`, `stdev()`, and `var()`. This returns the following table:

10 results over all time

Options... Results per page 10

Overlay: None

	Region ↕	mean(Magnitude) ↕	stdev(Magnitude) ↕	var(Magnitude) ↕
1	Central California	1.368750	0.402867	0.162302
2	Greater Los Angeles area, California	1.450000	0.433799	0.188182
3	Gulf of California	4.200000	0.000000	0.000000
4	Gulf of Santa Catalina, California	1.900000	0.000000	0.000000
5	Long Valley area, California	1.237500	0.304432	0.092679
6	Northern California	1.506863	0.373572	0.139556
7	San Francisco Bay area, California	1.425000	0.331464	0.109868
8	Southern California	1.415000	0.378331	0.143134
9	offshore Central California	1.850000	0.393700	0.155000
10	offshore Northern California	1.800000	0.576194	0.332000

The `mean` values should be exactly the same as the values calculated using `avg()` in Example 2.

Example 4

This example uses the sample dataset from the 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 get the sample data into Splunk.
- Download the CSV file from **Use field lookups tutorial** and follow the instructions to set up your field lookup.

The original data set includes a `product_id` field that is the catalog number for the items sold at the Flower & Gift shop. The field lookup adds three new fields to your events: `product_name`, which is a descriptive name for the item; `product_type`, which is a category for the item; and `price`, which is the cost of the item.

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 Flower & Gift shop by their ID, type, and name. Also, calculate the revenue for each product.

```
sourcetype=access_* action=purchase | stats values(product_type) AS  
Type, values(product_name) AS Name, sum(price) AS "Revenue" by  
product_id | rename product_id AS "Product ID" | eval Revenue="$  
".toString(Revenue,"commas")
```

This example uses the `values()` function to display the corresponding `product_type` and `product_name` values for each `product_id`. 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:

9 results over all time

Options... Results per page 20

Overlay: None

	Product ID ↕	Type ↕	Name ↕	Revenue ↕
1	AV-CB-01	BALLOONS	Birthday Wishes Balloons	\$ 9,425
2	FI-FW-02	GIFTS	Bountiful Fruit Basket	\$ 14,118
3	FI-SW-01	GIFTS	Tea & Spa Gift Set	\$ 31,595
4	FL-DLH-02	PLANTS	Gardenia Bonsai Plant	\$ 27,571
5	FL-DSH-01	FLOWERS	Dreams of Lavender Bouquet	\$ 16,905
6	K9-BD-01	FLOWERS	Vibrant Countryside Bouquet	\$ 20,709
7	K9-CW-01	FLOWERS	Beloved's Embrace Bouquet	\$ 33,957
8	RP-LI-02	CANDY	Decadent Chocolate Assortment	\$ 21,122
9	RP-SN-01	PLANTS	Fragrant Jasmine Plant	\$ 31,581

It looks like the top 3 purchases over the course of the week were the Beloved's Embrace Bouquet, the Tea & Spa Gift Set, and the Fragrant Jasmine Plant.

Example 5

This example uses generated email data (`sourcetype=cisco_esa`). You should be able to run this example on any email data by replacing the `sourcetype=cisco_esa` with your data's `sourcetype` value and the `mailfrom` field with your data's email address field name (for example, it might be `To`, `From`, or `Cc`).

Find out how much of your organization's email comes from com/net/org or other top level domains.

```
sourcetype="cisco_esa" mailfrom=* | eval
accountname=split(mailfrom,"@") | eval
from_domain=mvindex(accountname,-1) | stats
count(eval(match(from_domain, "[^\\n\\r\\s]+\\.com"))) AS ".com",
count(eval(match(from_domain, "[^\\n\\r\\s]+\\.net"))) AS ".net",
count(eval(match(from_domain, "[^\\n\\r\\s]+\\.org"))) AS ".org",
count(eval(NOT match(from_domain, "[^\\n\\r\\s]+\\.com|net|org"))) AS
"other"
```

The first half of this search uses `eval` to break up the email address in the `mailfrom` field and define the `from_domain` as the portion of the `mailfrom` field after the `@` symbol.

The results are then piped into the `stats` command. The `count()` function is used to count the results of the `eval` expression. Here, `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`.

This produces the following results table:

1 results in the last 60 minutes (from 3:38:00 PM to 4:38:03 PM on Monday, October 4, 2010)

Options... Results per page 10

Overlay: None

	.com	.net	.org	other
1	454	41	5	19

Example 6

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 **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Search Web access logs, and return the total number of hits from the top 10 referring domains. (The "top" command returns a count and percent value for each `referer`.)

```
sourcetype=access_* | top limit=10 referer | stats sum(count) AS total
```

This search uses the `top` command to find the ten most common referer

domains, which are values of the `referer` field. (You might also see this as `referer_domain`.) The results of `top` are then piped into the `stats` command. This example uses the `sum()` function to add the number of times each `referer` accesses the website. This summation is then saved into a field, `total`. This produces the single numeric value:



1 results yesterday (during Sunday, October 3, 2010)

Overlay: None

	total
1	483

More examples

Example 1: 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
```

Example 2: 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
```

Example 3: Remove duplicates of results with the same "host" value and return the total count of the remaining results.

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

Example 4: Return the average transfer rate for each host.

```
sourcetype=access* | stats avg(kbps) by host
```

See also

[eventstats](#), [rare](#), [sistats](#), [streamstats](#), [top](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the stats command.

strcat

Synopsis

Concatenates string values.

Syntax

strcat [allrequired=<bool>] <srcfields>* <destfield>

Required arguments

<destfield>

Syntax: <string>

Description: A destination field to save the concatenated string values defined by srcfields. The destfield is always at the end of the series of srcfields.

<srcfields>

Syntax: (<field>|<quoted-str>)

Description: Specify either key names or quoted literals.

quoted-str

Syntax: "<string>"

Description: Quoted literals.

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. By default, allrequired=f, meaning that 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.

Description

Stitch together fields and/or strings to create a new field. Quoted tokens are assumed to be literals and the rest field names. The destination field name is always at the end.

Examples

Example 1: Add the field, `comboIP`, which combines the source and destination IP addresses and separates them with a front slash character.

```
... | strcat sourceIP "/" destIP comboIP
```

Example 2: Add the field, `comboIP`, and then create a chart of the number of occurrences of the field values.

```
host="mailserver" | strcat sourceIP "/" destIP comboIP | chart count by  
comboIP
```

Example 3: Add a field, `address`, which combines the host and port values into the format `<host>::<port>`.

```
... | 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

The `streamstats` command, similar to the [stats](#) command, calculates summary statistics on search results. Unlike, `stats` (which works on the results as a whole), `streamstats` calculates statistics for each event at the time the event is seen.

Synopsis

Adds summary statistics to all search results in a streaming manner.

Syntax

```
streamstats [current=<bool>] [window=<int>] [global=<bool>] [allnum=<bool>]  
<stats-agg-term>* [<by clause>]
```

Required arguments

stats-agg-term

Syntax: <stats-func>(<evaluated-field> | <wc-field>) [AS <wc-field>]

Description: A statistical specifier optionally renamed to a new field name. The specifier can be by an aggregation function applied to a field or set of fields or an aggregation function applied to an arbitrary eval expression.

Optional arguments

current

Syntax: current=<bool>

Description: If true, tells Splunk to include the given, or current, event in the summary calculations. Defaults to true.

window

Syntax: window=<int>

Description: The 'window' option specify window size to be used in computing the statistics. Defaults to 0, which means that all previous (plus current) events are used.

global

Syntax: global=<bool>

Description: If the 'global' option is set to false and 'window' is set to a non-zero value, a separate window is used for each group of values of the group by fields. Defaults to true.

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. Defaults to false.

by clause

Syntax: by <field-list>

Description: The name of one or more fields to group by.

Stats functions options

stats-function

Syntax: avg() | c() | count() | dc() | distinct_count() | first() | last() | list() | max() | median() | min() | mode() | p<in>() | perc<int>() | per_day() | per_hour() | per_minute() | per_second() | range() | stdev() | stdevp() |

`sum() | sumsq() | values() | var() | varp()`

Description: Functions used with the stats command. Each time you invoke the `stats` command, you can use more than one function; however, you can only use one `by` clause. For a list of stats functions with descriptions and examples, see "[Functions for stats, chart, and timechart](#)".

Description

The `streamstats` command is similar to the `eventstats` command except that it uses events before a given event to compute the aggregate statistics applied to each event. If you want to include the given event in the stats calculations, use `current=true` (which is the default).

Example 1

Each day you track unique users, and you'd 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`.

Example 2

This example uses `streamstats` to produce hourly cumulative totals for category values.

```
... | timechart span=1h sum(value) as total by category | streamstats  
global=f sum(total) as accu_total
```

The `timechart` command buckets the events into spans of 1 hour and counts the total values for each category. The `timechart` command will also fill NULL values, so that there are no missing values. Then, the `streamstats` command is used to calculate the accumulated total.

More examples

Example 1: Compute the average value of foo for each value of bar including only the only 5 events with that value of bar.

```
... | streamstats avg(foo) by bar window=5 global=f
```

Example 2: 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
```

Example 3: 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, etc.

```
... | streamstats count
```

If you didn't want to include the current event, you would specify:

```
... | streamstats count current=f
```

See also

[accum](#), [autoregress](#), [delta](#), [fillnull](#), [eventstats](#), [stats](#), [streamstats](#), [trendline](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `streamstats` command.

table

The `table` command is similar to the `fields` command in that it enables you to specify the fields you want to keep in your results. Use `table` command when you want to retain data purely as a table.

The `table` command can be used to build a scatter plot to show trends in the relationships between discrete values of your data. Otherwise, you should not use it for charts (such as `chart` or `timechart`) because the UI requires the internal fields (which are the fields beginning with an underscore, `_*`) to render the charts, and the `table` command strips these fields out of the results by default. Instead, you should use the `fields` command because it always retains all the internal fields.

Synopsis

Creates a table using only the field names specified.

Syntax

table <wc-field-list>

Arguments

<wc-field-list>

Syntax: <wc-field> <wc-field> ...

Description: A list of field names, can include wildcards.

Description

The `table` command returns a table formed by only the fields specified 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 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`.

Examples

Example 1

This example uses recent (October 11-18, 2010) earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains the source network (Src), ID (Eqid), version, date, location, magnitude, depth (km) and number of reporting stations (NST) for each earthquake over the last 7 days.

Download the text file, **M 2.5+ earthquakes, past 7 days**, save it as a CSV file, and upload it to Splunk. Splunk should extract the fields automatically. Note that you'll be seeing data from the 7 days previous to your download, so your results will vary from the ones displayed below.

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`).

```
source="eqs7day-M1.csv" Region=*California | table Datetime, Region,
Magnitude, Depth
```

This simply reformats your events into a table and displays only the fields that you specified as arguments.

57 results yesterday (during Sunday, October 17, 2010)

« prev 1 2 3 4 5 6 next » | Options... Results per page 10

Overlay: None

	Datetime ↕	Region ↕	Magnitude ↕	Depth ↕
1	Monday, October 18, 2010 06:48:57 UTC	Northern California	1.1	3.10
2	Monday, October 18, 2010 06:41:00 UTC	Central California	2.2	16.50
3	Monday, October 18, 2010 06:40:40 UTC	Northern California	2.0	7.50
4	Monday, October 18, 2010 06:39:27 UTC	Southern California	1.3	11.30
5	Monday, October 18, 2010 05:20:15 UTC	Southern California	1.1	12.00
6	Monday, October 18, 2010 05:18:11 UTC	Southern California	2.1	8.20
7	Monday, October 18, 2010 05:16:43 UTC	Central California	1.1	4.20
8	Monday, October 18, 2010 03:44:43 UTC	Northern California	3.1	0.00
9	Monday, October 18, 2010 03:05:39 UTC	Southern California	1.7	13.10
10	Monday, October 18, 2010 03:02:35 UTC	Southern California	1.5	12.50

Example 2

This example uses recent (October 11-18, 2010) earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains the source network (Src), ID (Eqid), version, date, location, magnitude, depth (km) and number of reporting stations (NST) for each earthquake over the last 7 days.

Download the text file, **M 2.5+ earthquakes, past 7 days**, save it as a CSV file, and upload it to Splunk. Splunk should extract the fields automatically. Note that you'll be seeing data from the 7 days previous to your download, so your results will vary from the ones displayed below.

Show the date, time, coordinates, and magnitude of each recent earthquake in Northern California.

```
source="eqs7day-M1.csv" Region="Northern California" | rename Lat AS
Latitude, Lon AS Longitude | table Datetime, L*, Magnitude
```

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 `L*`, the magnitude with `Magnitude`, and the date and time with `Datetime`.

Overlay:

	Datetime ↕	Magnitude ↕	Latitude ↕	Longitude ↕
1	Tuesday, October 19, 2010 18:46:47 UTC	1.2	38.7947	-122.7583
2	Tuesday, October 19, 2010 17:57:13 UTC	1.2	38.8393	-122.7727
3	Tuesday, October 19, 2010 17:56:42 UTC	1.2	38.8385	-122.7892
4	Tuesday, October 19, 2010 17:51:48 UTC	3.4	38.8502	-122.7962
5	Tuesday, October 19, 2010 17:48:31 UTC	1.2	38.8262	-122.7965
6	Tuesday, October 19, 2010 17:29:44 UTC	1.4	38.5233	-122.6067
7	Tuesday, October 19, 2010 11:57:19 UTC	1.8	41.8817	-122.2733
8	Tuesday, October 19, 2010 11:42:56 UTC	1.6	38.8492	-122.8192
9	Tuesday, October 19, 2010 11:23:51 UTC	1.1	38.8233	-122.8055
10	Tuesday, October 19, 2010 11:23:51 UTC	1.2	38.8245	-122.8063

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 Splunk. 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`):

Overlay:

	clientip ↕	network ↕
1	192.0.1.51	local
2	192.168.11.33	other
3	192.168.11.44	other
4	192.168.11.35	other
5	192.1.2.40	other
6	192.1.2.35	other
7	192.0.1.39	local
8	192.1.2.52	other
9	192.1.2.39	other
10	192.0.1.30	local

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

Synopsis

Annotates specified fields in your search results with tags.

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. Defaults to F.

inclvalue

Syntax: inclvalue=T|F

Description: If outputfield is specified, controls whether or not the field value is added to the output field. Defaults to F.

Description

Annotate the 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 .

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

Synopsis

Returns the last `n` number of specified results.

Syntax

`tail [<N>]`

Required arguments

`<N>`

Syntax: `<int>`

Description: The number of results to return, default is 10 if none is specified.

Description

Returns the last `n` results, or 10 if no integer is specified. The events are returned in reverse order, starting at the end of the result set.

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

Synopsis

Creates a time series chart with corresponding table of statistics.

Syntax

```
timechart [sep=<string>] [partial=<bool>] [cont=<t|f>] [limit=<int>]  
[agg=<stats-agg-term>] [<bucketing-option>]* (<single-agg> [by  
<split-by-clause>]) | ( (<eval-expression>) by <split-by-clause> )
```

Required arguments

agg

Syntax: <stats-agg-term>

Description: See the Stats functions section below. For a list of stats functions with descriptions and examples, see ["Functions for stats, chart, and timechart"](#).

bucketing option

Syntax: bins | minspan | span | <start-end>

Description: Discretization options. If a bucketing option is not supplied, `timechart` defaults to `bins=100`. `bins` sets the maximum number of bins, not the target number of bins.

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. The following are the basic operations you can perform with eval. 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 may not produce valid results if the values are not numerical. Additionally, Splunk can concatenate the two operands if they are both strings. When concatenating values with '.',

Splunk treats both values as strings regardless of their actual type.

single-agg

Syntax: count|<stats-func>(<field>)

Description: A single aggregation applied to a single field (can be eval'd field). No wildcards are allowed. The field must be specified, except when using the special 'count' aggregator that applies to events as a whole.

split-by-clause

Syntax: <field> (<tc-option>)* [<where-clause>]

Description: Specifies a field to split by. If field is numerical, default discretization is applied; discretization is defined with tc-option.

Optional arguments

cont

Syntax: cont=<bool>

Description: Specifies whether the chart is continuous or not. If true, Splunk fills in the time gaps. Defaults is True|T.

fixedrange

Syntax: fixedrange=<bool>

Description: (Not valid for 4.2) Specify whether or not to enforce the earliest and latest times of the search. Setting it to false allows the timechart to constrict to just the time range with valid data. Default is True|T.

limit

Syntax: limit=<int>

Description: Specify a limit for series filtering; limit=0 means no filtering. By default, setting limit=N would filter the top N values based on the sum of each series.

partial

Syntax: partial=<bool>

Description: Controls if partial time buckets should be retained or not. Only the first and last bucket could ever be partial. Defaults to True|T, meaning that they are retained.

sep

Syntax: sep=<string>

Description: Specifies the separator to use for output fieldnames when multiple data series are specified along with a split-by field.

Stats functions

stats-agg-term

Syntax: <stats-func>(<evaluated-field> | <wc-field>) [AS <wc-field>]

Description: A statistical specifier optionally renamed to a new field name. The specifier can be by an aggregation function applied to a field or set of fields or an aggregation function applied to an arbitrary eval expression.

stats-function

Syntax: avg() | c() | count() | dc() | distinct_count() | earliest() | estdc() | estdc_error() | exactperc<int>() | first() | last() | latest() | list() | max() | median() | min() | mode() | p<int>() | perc<int>() | per_day() | per_hour() | per_minute() | per_second() | range() | stdev() | stdevp() | sum() | sumsq() | upperperc<int>() | values() | var() | varp()

Description: Functions used with the stats command. Each time you invoke the `stats` command, you can use more than one function; however, you can only use one `by` clause. For a list of stats functions with descriptions and examples, see ["Functions for stats, chart, and timechart"](#).

Bucketing 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 bucket size that results in no more than 100 distinct buckets. Even though you specify 100 or 300, the resulting number of buckets might be much lower.) Defaults to 100.

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=<span-length>

Description: Sets the size of each bucket, 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 buckets. Data outside of the [start, end] range is discarded.

Log span syntax

<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).

Span length syntax

span-length

Syntax: [<timescale>]

Description: A span length based on time.

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.

<sec>

Syntax: s | sec | secs | second | seconds

Description: Time scale in seconds.

<min>

Syntax: m | min | mins | minute | minutes

Description: Time scale in minutes.

<hr>

Syntax: h | hr | hrs | hour | hours

Description: Time scale in hours.

<day>

Syntax: d | day | days

Description: Time scale in days.

<month>

Syntax: mon | month | months

Description: Time scale in months.

<subseconds>

Syntax: us | ms | cs | ds

Description: Time scale in microseconds (us), milliseconds (ms), centiseconds (cs), or deciseconds (ds).

tc options

tc-option

Syntax: <bucketing-option> | usenull=<bool> | useother=<bool> | nullstr=<string> | otherstr=<string>

Description: Options for controlling the behavior of splitting by a field.

usenull

Syntax: usenull=<bool>

Description: Controls whether or not a series is created for events that do not contain the split-by field.

nullstr

Syntax: nullstr=<string>

Description: If usenull is true, this series is labeled by the value of the nullstr option. Defaults to NULL.

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>. Defaults to True|T.

otherstr

Syntax: otherstr=<string>

Description: If useother is true, this series is labeled by the value of the otherstr option. Defaults to OTHER.

where clause

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.

<where-comp>

Syntax: <wherein-comp> | <wherethresh-comp>

Description: A criteria for the where clause.

<wherein-comp>

Syntax: (in|notin) (top|bottom)<int>

Description: A where-clause criteria that requires the aggregated series value be in or not in some top or bottom grouping.

<wherethresh-comp>

Syntax: (<|>)()?<num>

Description: A where-clause criteria that requires the aggregated series value be greater than or less than some numeric threshold.

Description

Create a chart for a statistical aggregation applied to a field against time as the x-axis. Data is optionally split by a field so that each distinct value of this split-by field is a series. If you use an eval expression, the split-by clause is required. The limit and agg options enables you to specify series filtering but are ignored if an explicit where-clause is provided (limit=0 means no series filtering).

Bucket 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.

A note about split-by fields

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:

```
... | chart sum(A) by A span=log2
```

However, you can work around this with an `eval` expression, for example:

```
... | eval A1=A | chart sum(A) by A1 span=log2
```

Examples

Example 1

This example uses the sample dataset from the tutorial and a field lookup to add more information to the event data.

- Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk.
- Download the CSV file from **this topic in the tutorial** and follow the instructions to set up your field lookup.

The original data set includes a `product_id` field that is the catalog number for the items sold at the Flower & Gift shop. The field lookup adds three new fields to your events: `product_name`, which is a descriptive name for the item; `product_type`, which is a category for the item; and `price`, which is the cost of the item.

After you configure the field lookup, you can run this search using the time range, **Other > Yesterday**.

Chart revenue for the different product that were purchased yesterday.

```
sourcetype=access_* action=purchase | timechart per_hour(price) by  
product_name usenull=f
```

This example searches for all purchase events (defined by the `action=purchase`) and pipes those results into the `timechart` command. The `per_hour()` function sums up the values of the `price` field for each item (`product_name`) and buckets the total for each hour of the day.

This produces the following table of results:

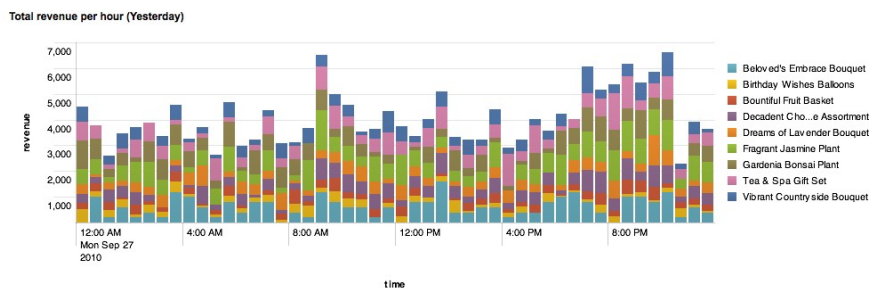
48 results yesterday (during Monday, September 27, 2010)

« prev 1 2 3 4 5 next » | Options... Results per page 10

Overlay: None

	_time	Beloved's Embrace Bouquet	Birthday Wishes Balloons	Bountiful Fruit Basket	Decadent Chocolate Assortment	Dreams of Lavender
1	9/27/10 12:00:00.000 AM		522.000000	234.000000	354.000000	392.000000
2	9/27/10 12:30:00.000 AM	990.000000	232.000000	312.000000	236.000000	98.000000
3	9/27/10 1:00:00.000 AM	198.000000	290.000000	312.000000	472.000000	294.000000
4	9/27/10 1:30:00.000 AM	594.000000	290.000000	78.000000	472.000000	196.000000
5	9/27/10 2:00:00.000 AM	198.000000	116.000000	390.000000	472.000000	392.000000

Click **Show report** to format the chart in Report Builder. Here, it's formatted as a stacked column chart over time:



After you create this chart, you can mouseover 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 cause is 24 hours).

Example 2

This example uses the sample dataset from the tutorial and a field lookup to add more information to the event data.

- Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk.
- Download the CSV file from **this topic in the tutorial** and follow the instructions to set up your field lookup.

The original data set includes a `product_id` field that is the catalog number for the items sold at the Flower & Gift shop. The field lookup adds three new fields to your events: `product_name`, which is a descriptive name for the item; `product_type`, which is a category for the item; and `price`, which is the cost of the item.

After you configure the field lookup, you can run this search using the time range, **All time**.

Chart the number of purchases made daily for each type of product.

```
sourcetype=access_* action=purchase | timechart span=1d count by product_type 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 tells Splunk to ignore any events that contain a NULL value for `product_type`. This produces the following table:

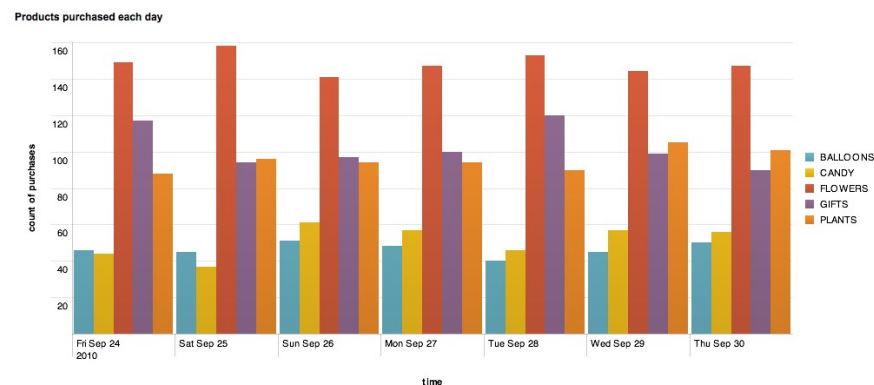
7 results over all time

Results per page: 10

Overlay: None

	time	BALLOONS	CANDY	FLOWERS	GIFTS	PLANTS
1	9/24/10 12:00:00.000 AM	46	44	149	117	88
2	9/25/10 12:00:00.000 AM	45	37	158	94	96
3	9/26/10 12:00:00.000 AM	51	61	141	97	94
4	9/27/10 12:00:00.000 AM	48	57	147	100	94
5	9/28/10 12:00:00.000 AM	40	46	153	120	90
6	9/29/10 12:00:00.000 AM	45	57	144	99	105
7	9/30/10 12:00:00.000 AM	50	56	147	90	101

Click **Show report** to format the chart in Report Builder. Here, it's formatted as a column chart over time:



You can compare the number of different items purchased each day and over the course of the week. It looks like day-to-day, the number of purchases for each item do not vary significantly.

Example 3

This example uses the sample dataset from the tutorial and a field lookup to add more information to the event data.

- Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk.
- Download the CSV file from **this topic in the tutorial** and follow the instructions to set up your field lookup.

The original data set includes a `product_id` field that is the catalog number for the items sold at the Flower & Gift shop. The field lookup adds three new fields to your events: `product_name`, which is a descriptive name for the item; `product_type`, which is a category for the item; and `price`, which is the cost of the item.

After you configure the field lookup, you can run this search using the time range, **All time**.

Count the total revenue made for each item sold at the shop over the course of the week. This examples shows two ways to do this.

1. This first search uses the `span` argument to bucket the times of the search results into 1 day increments. Then uses the `sum()` function to add the `price` for each `product_name`.

```
sourcetype=access_* action=purchase | timechart span=1d sum(price) by product_name usenull=f
```

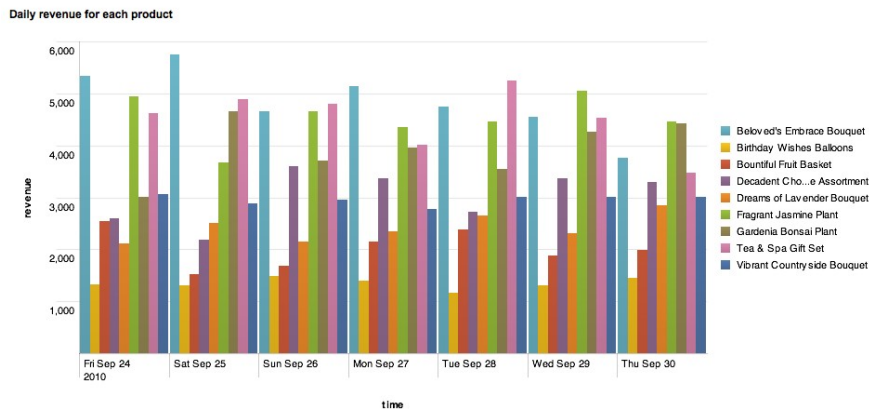
2. This second search uses the `per_day()` function to calculate the total of the `price` values for each day.

```
sourcetype=access_* action=purchase | timechart per_day(price) by product_name usenull=f
```

Both searches produce the following results table:

7 results over all time					
Overlay: None		Results per page 10			
	<u>time</u>	Beloved's Embrace Bouquet	Birthday Wishes Balloons	Bountiful Fruit Basket	Decadent Chocolate Assortment
1	9/24/10 12:00:00.000 AM	5346	1334	2535	2596
2	9/25/10 12:00:00.000 AM	5742	1305	1521	2183
3	9/26/10 12:00:00.000 AM	4653	1479	1677	3599
4	9/27/10 12:00:00.000 AM	5148	1392	2145	3363
5	9/28/10 12:00:00.000 AM	4752	1160	2379	2714
6	9/29/10 12:00:00.000 AM	4554	1305	1872	3363
7	9/30/10 12:00:00.000 AM	3762	1450	1989	3304

Click **Show report** to format the chart in Report Builder. Here, it's formatted as a column chart over time:



Now you can compare the total revenue made for items purchased each day and over the course of the week.

Example 4

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Chart yesterday's views and purchases at the Flower & Gift shop.

```
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. This produces the following results table:

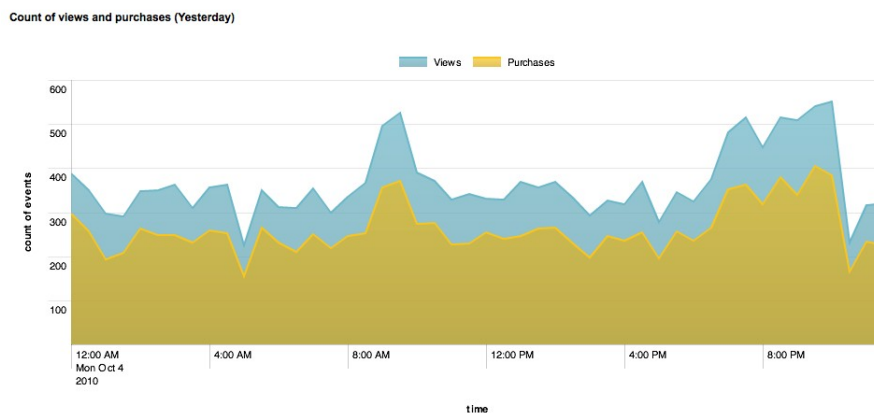
48 results yesterday (during Saturday, October 2, 2010)

« prev 1 2 3 next » | Options... Results per page 20

Overlay: None

	time ↕	Views ↕	Purchases ↕
1	10/2/10 12:00:00.000 AM	388.000000	296.000000
2	10/2/10 12:30:00.000 AM	350.000000	256.000000
3	10/2/10 1:00:00.000 AM	296.000000	192.000000
4	10/2/10 1:30:00.000 AM	290.000000	208.000000
5	10/2/10 2:00:00.000 AM	348.000000	262.000000
6	10/2/10 2:30:00.000 AM	350.000000	248.000000
7	10/2/10 3:00:00.000 AM	362.000000	248.000000

Click **Show report** to format the chart in Report Builder. Here, it's formatted as an area chart:



The difference between the two areas indicates that all the views did not lead to purchases. If all views lead to purchases, you would expect the areas to overlay atop each other completely so that there is no difference between the two areas.

Example 5

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 **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Search the Web access logs and count the number of page requests over time.

```
sourcetype=access_* | timechart count(eval(method="GET")) AS GET,
count(eval(method="POST")) AS POST
```

This search uses the `count()` function and `eval` expressions to count the different page request methods, `GET` or `POST`. This produces the following result table:

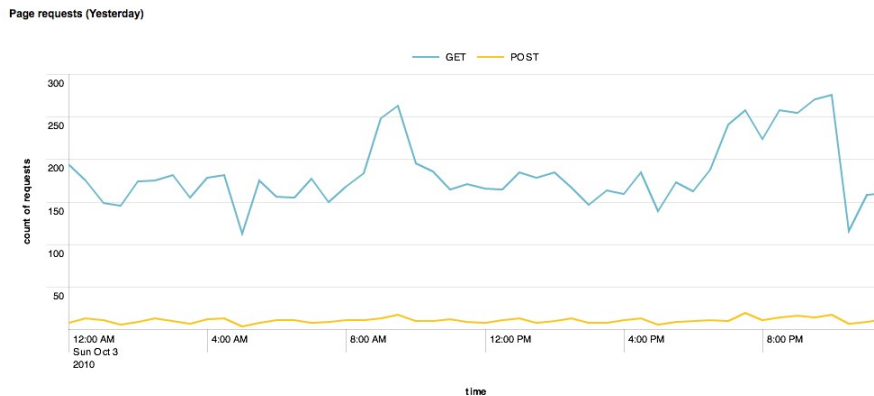
48 results yesterday (during Sunday, October 3, 2010)

« prev 1 2 3 next » | Options... Results per page 20

Overlay: None

	time	GET	POST
1	10/3/10 12:00:00.000 AM	194	7
2	10/3/10 12:30:00.000 AM	175	13
3	10/3/10 1:00:00.000 AM	148	11
4	10/3/10 1:30:00.000 AM	145	5
5	10/3/10 2:00:00.000 AM	174	9
6	10/3/10 2:30:00.000 AM	175	13
7	10/3/10 3:00:00.000 AM	181	10

Click **Show report** to format the chart in Report Builder. Here, it's formatted as a line chart:



Note: You can use the `stats`, `chart`, and `timechart` commands to perform the same statistical calculations on your data. The `stats` command returns a table of results. The `chart` command returns the same table of results, but you can use the Report Builder to format this table as a chart. If you want to chart your results over a time range, use the `timechart` command. You can also see variations of this example with the [chart](#) and [timechart](#) commands.

More examples

Example 1: Compute the product of the average "CPU" and average "MEM" each minute for each "host"


```
... | timechart span=1m eval(avg(CPU) * avg(MEM)) by host
```

Example 2: Display timechart of the avg of cpu_seconds by processor rounded to 2 decimal places.

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

Example 3: Calculate the average value of "CPU" each minute for each "host".

```
... | timechart span=1m avg(CPU) by host
```

Example 4: Create a timechart of average "cpu_seconds" by "host", and remove data (outlying values) that may distort the timechart's axis.

```
... | timechart avg(cpu_seconds) by host | outlier action=tf
```

Example 5: Graph the average "thruput" of hosts over time.

```
... | timechart span=5m avg(thruput) by host
```

Example 6: Example usage

```
sshd failed OR failure | timechart span=1m count(eventtype) by source_ip  
usenull=f where count>10
```

See also

[bucket](#), [chart](#), [sitimechart](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the timechart command.

top

Synopsis

Displays the most common values of a field.

Syntax

```
top <top-opt>* <field-list> [<by-clause>]
```

Required arguments

<field-list>

Syntax: <field>, ...

Description: Comma-delimited list of field names.

<top-opt>

Syntax: countfield=<string> | limit=<int> | otherstr=<string> |
percentfield=<string> | showcount=<bool> | showperc=<bool> |
useother=<bool>

Description: Options for top.

Optional arguments

<by-clause>

Syntax: by <field-list>

Description: The name of one or more fields to group by.

Top options

countfield

Syntax: countfield=<string>

Description: Name of a new field to write the value of count, default is "count".

limit

Syntax: limit=<int>

Description: Specifies how many tuples to return, "0" returns all values. Default is "10".

otherstr

Syntax: otherstr=<string>

Description: If useother is true, specify the value that is written into the row representing all other values. Default is "OTHER".

percentfield

Syntax: percentfield=<string>

Description: Name of a new field to write the value of percentage, default is "percent".

showcount

Syntax: showcount=<bool>

Description: Specify whether to create a field called "count" (see "countfield" option) with the count of that tuple. Default is 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 is 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 is false.

Description

Finds the most frequent tuple of values of all fields in the field list, along with a count and percentage. If a the optional by-clause is provided, we will find the most frequent values for each distinct tuple of values of the group-by fields.

Examples

Example 1: Return the 20 most common values of the "url" field.

```
... | top limit=20 url
```

Example 2: Return top "user" values for each "host".

```
... | top user by host
```

Example 3: Return top URL values.

```
... | top url
```

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

Synopsis

Groups events into transactions.

Syntax

transaction [<field-list>] [name=<transaction-name>] [<txn_definition-opt>]*
[<memcontrol-opt>]* [<rendering-opt>]*

Required arguments

txn_definition-opt

Syntax: <maxspan> | <maxpause> | <maxevents> | <startswith> |
<endswith> | <connected> | <unifyends> | <keeporphans>

Description: Transaction definition options.

memcontrol-opt

Syntax: <maxopentxn> | <maxopenevents> | <keepevicted>

Description: Memory constraint options.

rendering-opt

Syntax: <delim> | <mvlist> | <mvraw> | <>nullstr>

Description: Multivalue rendering options.

Optional arguments

field-list

Syntax: <string>, ...

Description: One field or a list of 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.

name

Syntax: name=<transaction-name>

Description: The name of a stanza from `transactiontypes.conf` to be used for finding transactions. If other arguments (e.g., maxspan) are provided, they overrule the value specified in the transaction definition.

Transaction definition options

connected=<bool>

Description: Relevant if fields is not empty. Controls whether an event that is not inconsistent and not consistent with the fields of a transaction, opens a new transaction (connected=t) or is added to the transaction. An event can be not inconsistent and not consistent if it contains fields required by the transaction but none of these fields has been instantiated in the transaction (by a previous event addition).

endswith=<filter-string>

Description: A search or eval filtering expression which if satisfied by an event marks the end of a transaction.

keeporphans=<bool>

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. Defaults to false|f.

maxspan=<int>(s|m|h|d)?

Description: The maxspan constraint requires the transaction's events to span less than maxspan. If value is negative, disable the maxspan constraint. By default, maxspan=-1 (no limit).

maxpause=<int>(s|m|h|d)?

Description: The maxpause constraint requires there be no pause between a transaction's events of greater than maxpause. If value is negative, disable the maxpause constraint. By default, maxpause=-1 (no limit).

maxevents=<int>

Description: The maximum number of events in a transaction. If the value is negative this constraint is disabled. By default, maxevents=1000.

startswith=<filter-string>

Description: A search or eval filtering expression which if satisfied by an event marks the beginning of a new transaction.

unifyends=<bool>

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. By default, unifyends=f.

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.

<quoted-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

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, which is set to '0' for evicted transactions and '1' for closed ones. 'closed_txn' is set to '1' if one of the following conditions is hit: maxevents, maxpause, maxspan, startswith (for this last one, because transaction 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 will be output even though all transactions will have 'closed_txn' set to '0'. A transaction can also be evicted when the memory limitations are reached.

maxopenevents=<int>

Description: Specifies the maximum number of events (which are) part of open transactions before transaction eviction starts happening, using LRU policy. The default value of this field is read from the transactions stanza in limits.conf.

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. The default value of this field is read from the

transactions stanza in limits.conf.

Multivalue rendering options

delim=<string>

Description: In conjunction with mvraw=t, a string used to delimit the values of `_raw`. By default, delim=" ".

mvlist=<bool> | <field-list>

Description: Flag controlling whether the multivalued fields of the transaction are (mvlist=t) a list of the original events ordered in arrival order or (mvlist=f) a set of unique field values ordered lexicographically. If a comma/space delimited list of fields is provided only those fields are rendered as lists. By default, mvlist=f.

mvraw=<bool>

Description: Used to specify whether the `_raw` field of the transaction search result should be a multivalued field. By default, mvraw=f.

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. By default, nullstr="NULL".

Description

Given events as input, 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.

Splunk 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, the `transaction` command will 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
```

The `transaction` command produces two fields, `duration` and `eventcount`. The `duration` value is the difference between the timestamps for the first and last events in the transaction. The `eventcount` value is the number of events in the transaction.

Examples

Example 1

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

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:

3,958 events yesterday (during Sunday, September 26, 2010)	
« prev	1 2 3 4 5 6 7 8 9 10 next » Options...
Results per page	10
1	9/26/10 11:59:34.000 PM
178.19.3.39 - - [26/Sep/2010:23:59:34] "GET /flower_store/images/cat3.gif HTTP/1.1" 200 5024 "http://mystore.splunk.com/flower_store/item_screen?item_id=EST-14&JSESSIONID=SD5SL10FF8ADFF3" Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3966 3244	
178.19.3.39 - - [26/Sep/2010:23:59:34] "GET /flower_store/category.screen?category_id=CANDY HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-14&JSESSIONID=SD5SL10FF8ADFF3" Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 842 96	
178.19.3.39 - - [26/Sep/2010:23:59:34] "GET /flower_store/category.screen?category_id=CANDY HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-14&JSESSIONID=SD5SL10FF8ADFF3" Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3187 1245	
host=apache2.splunk.com host=apache3.splunk.com sourcetype=access_combined_wookiee source=Sampledata.zip:apache2.splunk.com/access_combined.log	
2	9/26/10 11:59:15.000 PM
10.192.1.46 - - [26/Sep/2010:23:59:15] "POST /flower_store/order.do HTTP/1.1" 200 13849 "http://mystore.splunk.com/flower_store/enter_order_information.screen&JSESSIONID=SD5SL10FF8ADFF3" Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 1463 2971	
10.192.1.46 - - [26/Sep/2010:23:59:15] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-27&JSESSIONID=SD5SL10FF8ADFF3" Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 2672 2340	
10.192.1.46 - - [26/Sep/2010:23:59:15] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-27&JSESSIONID=SD5SL10FF8ADFF3" Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 176 4680	
10.192.1.46 - - [26/Sep/2010:23:59:15] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-27&JSESSIONID=SD5SL10FF8ADFF3" Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3352 1920	
host=apache1.splunk.com host=apache3.splunk.com sourcetype=access_combined_wookiee source=Sampledata.zip:apache1.splunk.com/access_combined.log source=Sampledata.zip:apache3.splunk.com/access_combined.log	

This search groups events together based on the IP addresses accessing the server and the time constraints. The search results may 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 were shopping from the same office. For more information, read the topic "About transactions" in the *Knowledge Manager manual*.

Example 2

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

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.

```
sourcetype=access_* | transaction clientip host maxspan=30s maxpause=5s
```

This produces the following events list:

7,050 events yesterday (during Sunday, September 26, 2010)	
1	9/26/10 11:59:34.000 PM [26/Sep/2010:23:59:34] "GET /flower_store/images/cat3.gif HTTP/1.1" 200 5024 "http://mystore.splunk.com/flower_store/item.screen?item_id=EST-14&JSESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3966 3244
2	9/26/10 11:59:34.000 PM [26/Sep/2010:23:59:34] "GET /flower_store/category.screen?category_id=CANDY HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-14&JSESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 842 96

In contrast to the transaction in Example 1, each of these events have a distinct combination of the IP address (`clientip` values) and `host` values within the limits of the time constraints. Thus, you should not see different values of `host` or `clientip` addresses among the events in a single transaction.

Example 3

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **Other > Yesterday**.

Define a purchase transaction as 3 events from one IP address which occur in a ten minute span of time.

```
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 three events.

This produces the following events list:

3,779 events yesterday (during Sunday, October 3, 2010)	
« prev	1 2 3 4 5 6 7 8 9 10 next » Options...
Results per page	20
1	10/3/10 11:59:34.000 PM 178.19.3.39 - [03/Oct/2010:23:59:34] "GET /flower_store/category.screen?category_id=CANDY HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-1487SESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 842 96
2 events	178.19.3.39 - [03/Oct/2010:23:59:34] "GET /flower_store/category.screen?category_id=CANDY HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-1487SESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3187 1245
	host=apache2.splunk.com host=apache3.splunk.com source=Sampledata.zip:/apache2.splunk.com/access_combined.log source=Sampledata.zip:/apache3.splunk.com/access_combined.log sourcetype=access_combined_wcookie
2	10/3/10 11:59:15.000 PM 10.192.1.46 - [03/Oct/2010:23:59:15] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-2787SESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 2672 2340
3 events	10.192.1.46 - [03/Oct/2010:23:59:15] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-2787SESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 176 4680
	10.192.1.46 - [03/Oct/2010:23:59:15] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" 200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-2787SESSIONID=SD5SL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 3352 1920
	host=apache3.splunk.com source=Sampledata.zip:/apache3.splunk.com/access_combined.log sourcetype=access_combined_wcookie

This above results show the same IP address appearing from different host domains.

Example 4

This example uses generated email data (`sourcetype=cisco_esa`). You should be able to run this example on any email data by replacing the `sourcetype=cisco_esa` with your data's `sourcetype` value.

Define an email transaction as a group of up to 10 events each containing the same value for the `mid` (message ID), `icid` (incoming connection ID), and `dcid` (delivery connection ID) and with the last event in the transaction containing a "Message done" string.

```
sourcetype="cisco_esa" | transaction mid dcid icid maxevents=10  
endswith="Message done"
```

This produces the following events list:

333 events in the last 15 minutes (from 5:08:00 PM to 5:23:14 PM on Thursday, October 7, 2010)		
	« prev	1 2 3 4 5 6 7 8 9 10 next » Options...
	Results per page	10 ▾
1	10/7/10 5:23:05.000 PM	2010-10-7 17:23:5 2009 Info: MID 246872 Subject 'I made it LOL oh LOL' 2010-10-7 17:23:6 2009 Info: MID 246872 matched all recipients for per recipient policy DEFAULT in the inbound table 2010-10-7 17:23:6 2009 Info: MID 246872 interim verdict using engine: CASE spam positive 2010-10-7 17:23:6 2009 Info: MID 246872 using engine: CASE spam positive 2010-10-7 17:23:6 2009 Info: ISQ: Tagging MID 246872 for quarantine 2010-10-7 17:23:6 2009 Info: MID 246872 interim AV verdict using Sophos CLEAN 2010-10-7 17:23:6 2009 Info: MID 246872 antivirus negative 2010-10-7 17:23:7 2009 Info: RPC Delivery start RCID 17381 MID 246872 to local IronPort Spam Quarantine 2010-10-7 17:23:7 2009 Info: ISQ: Quarantined MID 246872 2010-10-7 17:23:7 2009 Info: RPC Message done RCID 17381 MID 246872 mid=246872 ▾
2	10/7/10 5:23:01.000 PM	2010-10-7 17:23:1 2009 Info: Start MID 246869 ICID 746151 2010-10-7 17:23:1 2009 Info: MID 246869 ICID 746151 From: <bounce birthday@mx.plaxo.com> 2010-10-7 17:23:1 2009 Info: MID 246869 ICID 746151 RID 0 To: <lennartvandenende@gmail.com> 2010-10-7 17:23:1 2009 Info: MID 246869 Message ID '<1257069161.1038946@plaxo.com>' 2010-10-7 17:23:2 2009 Info: MID 246869 ready 11485 bytes from <bounce birthday@mx.plaxo.com> 2010-10-7 17:23:2 2009 Info: MID 246869 matched all recipients for per recipient policy DEFAULT in the outbound table 2010-10-7 17:23:2 2009 Info: MID 246869 interim AV verdict using Sophos CLEAN 2010-10-7 17:23:2 2009 Info: MID 246869 antivirus negative 2010-10-7 17:23:3 2009 Info: MID 246869 rewritten to MID 246871 by add footer filter 'Footer Stamping' 2010-10-7 17:23:3 2009 Info: Message finished MID 246869 done icid=746151 ▾ mid=246869 ▾

Here, you can see that each transaction has no more than ten events. Also, the last event includes the string, "Message done" in the event line.

Example 5

This example uses generated email data (`sourcetype=cisco_esa`). You should be able to run this example on any email data by replacing the `sourcetype=cisco_esa` with your data's `sourcetype` value.

Define an email transaction as a group of up to 10 events each containing 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 five seconds apart and each transaction should have no more than ten events.

```
sourcetype="cisco_esa" | transaction mid dcid icid maxevents=10
maxspan=5s mvlist=t
```

By default, the values of multivalue fields are suppressed in search results (`mvlist=f`). Specifying `mvlist=t` in this search tells Splunk to display all the values of the selected fields. This produces the following events list:

516 events in the last 15 minutes (from 7:01:00 PM to 7:16:23 PM on Monday, October 4, 2010)

« prev 1 2 3 4 5 6 7 8 9 10 next » | Options... Results per page 10 ▾

1	10/4/10 7:16:22.000 PM	2010-10-4 19:16:22 2009 Info: New SMTP ICID 744566 Interface Management (192.168.3.120) address 206.176.229.254 reverse dns host ironport.mineralore.com verified yes 2010-10-4 19:16:22 2009 Info: ICID 744566 ACCEPT SG WHITELIST match 206.176.229.254 SBRS 5.1 host=ironport.mail.acme host=ironport.mail.acme sourcetype=cisco_esa sourcetype=cisco_esa source=/var/log/cisco_ironport_mail.log source=/var/log/cisco_ironport_mail.log duration=0
2	10/4/10 7:16:11.000 PM	2010-10-4 19:16:14 2009 Info: New SMTP ICID 744565 Interface Management (192.168.3.120) address 89.118.163.57 reverse dns host 89.118.163.57 static.albacom.net verified no 2010-10-4 19:16:14 2009 Info: ICID 744565 REJECT SG BLACKLIST match sbrs[10.0: 3.0] SBRS 9.0 host=ironport.mail.acme host=ironport.mail.acme sourcetype=cisco_esa sourcetype=cisco_esa source=/var/log/cisco_ironport_mail.log source=/var/log/cisco_ironport_mail.log duration=3
3	10/4/10 7:16:11.000 PM	2010-10-4 19:16:13 2009 Info: New SMTP ICID 744564 Interface Management (192.168.3.120) address 219.130.164.36 reverse dns host 36.164.130.219.broad.jm.gd.dynamic.163data.com.cn verified no 2010-10-4 19:16:13 2009 Info: ICID 744564 REJECT SG BLACKLIST match sbrs[10.0: 3.0] SBRS 10.0 2010-10-4 19:16:13 2009 Info: ICID 744564 close host=ironport.mail.acme host=ironport.mail.acme host=ironport.mail.acme sourcetype=cisco_esa sourcetype=cisco_esa source=/var/log/cisco_ironport_mail.log source=/var/log/cisco_ironport_mail.log source=/var/log/cisco_ironport_mail.log source=/var/log/cisco_ironport_mail.log duration=2

Here you can see that each transaction has a duration that is less than five seconds. Also, if there is more than one value for a field, each of the values is listed.

Example 6

This example uses the sample dataset from the tutorial. Download the data set from **this topic in the tutorial** and follow the instructions to upload it to Splunk. Then, run this search using the time range, **All time**.

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, "signon", and the last event contains the string, "purchase".

```
sourcetype=access_* | transaction JSESSIONID clientip
startswith="*signon*" endswith="purchase" | where duration>0
```

The search defines the first event in the transaction as events that include the string, "signon", using the `startswith="*signon*"` 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 the transactions that took less than a second to complete:

```

1 10/18/10 10.192.1.46 - - [18/Oct/2010:23:00:35] "POST /flower_store/j_signon_check HTTP/1.1" 302 309
11:00:35.000 PM "http://mystore.splunk.com/flower_store/enter_order_information.screen?JSESSIONID=SDSSL10FF8ADFF3" "Mozilla/5.0
(X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 427 3367
10.192.1.46 - - [18/Oct/2010:23:59:15] "POST /flower_store/order.do HTTP/1.1" 200 13849 "http://mystore.splunk.com
/flower_store/enter_order_information.screen?JSESSIONID=SDSSL10FF8ADFF3" "Mozilla/5.0 (X11; U; Linux i686; en-US;
rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 1463 2971
10.192.1.46 - - [18/Oct/2010:23:59:15] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" 200 10567
"http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-27&JSESSIONID=SDSSL10FF8ADFF3"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos
Firefox/1.5.0.10" 2672 2340
sourcetype=access_combined_wcookie | JSESSIONID=SDSSL10FF8ADFF3 | clientip=10.192.1.46 | duration=3520

2 10/18/10 192.168.11.40 - - [18/Oct/2010:21:45:42] "GET /flower_store/category.screen?category_id=FLOWERS HTTP/1.1" 200 10567
9:45:42.000 PM "http://mystore.splunk.com/flower_store/signon_welcome.screen?JSESSIONID=SDSSL1FF1ADFF4" "Mozilla/5.0 (X11; U;
Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/1.5.0.10" 4838 2187
192.168.11.40 - - [18/Oct/2010:21:47:45] "GET /flower_store/cart.do?action=purchase&itemId=EST-12 HTTP/1.1" 200
12399 "http://mystore.splunk.com/flower_store/product.screen?product_id=FL-DSH-01&JSESSIONID=SDSSL1FF1ADFF4"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos
Firefox/1.5.0.10" 3477 3385
sourcetype=access_combined_wcookie | JSESSIONID=SDSSL1FF1ADFF4 | clientip=192.168.11.40 | duration=123

```

You might be curious about why the transactions took a long time, so viewing these events may help you to troubleshoot. You won't see it in this data, but some transactions may take a long time because the user is updating and removing items from his shopping cart before he completes the purchase.

More examples

Example 1: 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
```

Example 2: 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
```

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the transaction command.

transpose

Synopsis

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]

Required arguments

int

Syntax: <int>

Description: Limit the number of rows to transpose. Default is 5.

Examples

Example 1: Transpose your first five search results, so that each column represents an event and each row, the field values.

```
... | 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

Synopsis

Computes the moving averages of fields.

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. Defaults to <trendtype><period>(<field>).

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.

Examples

Example 1: Computes a five event simple moving average for field 'foo' and write to new field 'smoothed_foo.' Also, in the same line, computes ten event exponential moving average for field 'bar' and write to field 'ema10(bar)'.

```
... | trendline sma5(foo) as smoothed_foo ema10(bar)
```

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.

typeahead

Synopsis

Returns typeahead on a specified prefix.

Syntax

typeahead prefix=<string> count=<int> [max_time=<int>] [<index-specifier>]
[<starttimeu>] [<endtimeu>] [*collapse*]

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 typeahead can run. If max_time=0, there is no limit.

starttimeu

Syntax: starttimeu=<int>

Description: Set the start time to N seconds since the epoch (Unix time). Defaults to 0.

endtimeu

Syntax: endtimeu=<int>

Description: Set the end time to N seconds since the epoch (Unix time). Defaults to now.

collapse

Syntax: collapse=<bool>

Description: Specify whether to collapse terms that are a prefix of another term and the event count is the same. Defaults to true.

Description

Returns typeahead on a specified prefix. Only returns a max of `count` results, can be targeted to an index and restricted by time.

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

Synopsis

Generates suggested eventtypes.

Syntax

typelearner [*grouping-field*] [*grouping-maxlen*]

Optional arguments

grouping-field

Syntax: <field>

Description: The field with values for `typelearner` to use when initially grouping events. Defaults to `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. Defaults to 15.

Description

Takes previous search results, and produces a list of promising searches that may be used as event-types. By default, the `typelearner` command initially groups events by the value of the grouping-field, and then further unifies and merges those groups, based on the keywords they contain.

Examples

Example 1: Have Splunk 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

Synopsis

Calculates the eventtypes for the search results

Syntax

`typer`

Description

Calculates the 'eventtype' field for search results that match a known event-type.

Examples

Example 1: Force Splunk to apply event types that you have configured (Splunk Web automatically does this when you view the "eventtype" field).

```
... | typer
```

See also

[typelearner](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `typer` command.

uniq

Synopsis

Filters out repeated adjacent results.

Syntax

```
uniq
```

Description

The `uniq` command works as a filter on the search results that you pass into it. It removes any search result if it is an exact duplicate with the previous result. This command does not take any arguments.

Note: We don't recommend running this command against a large dataset.

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

Synopsis

Converts results from a tabular format to a format similar to stats output. Inverse of `xyseries`.

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.

Examples

Example 1: Reformat the search results.

```
... | timechart avg(delay) by host | untable _time host avg_delay
```

See also

[xyseries](#)

where

Synopsis

Runs an eval expression to filter the results. The result of the expression must be Boolean.

Syntax

where <eval-expression>

Functions

The where command includes the following functions: `abs()`, `case()`, `ceil()`, `ceiling()`, `cidrmatch()`, `coalesce()`, `commands()`, `exact()`, `exp()`, `floor()`, `if()`, `ifnull()`, `isbool()`, `isint()`, `isnotnull()`, `isnull()`, `isnum()`, `isstr()`, `len()`, `like()`, `ln()`, `log()`, `lower()`, `ltrim()`, `match()`, `max()`, `md5()`, `min()`, `mvappend()`, `mvcount()`, `mvindex()`, `mvfilter()`, `mvjoin()`, `now()`, `null()`, `nullif()`, `pi()`, `pow()`, `random()`, `relative_time()`, `replace()`, `round()`, `rtrim()`, `searchmatch()`, `split()`, `sqrt()`, `strftime()`, `strptime()`, `substr()`, `time()`, `tonumber()`, `tostring()`, `trim()`, `typeof()`, `upper()`, `urldecode()`, `validate()`.

For **descriptions and examples** of each function, see "[Functions for eval and where](#)".

Description

The `where` command uses `eval` expressions to filter search results; it keeps only the results for which the evaluation was successful (that is, the Boolean result was true).

The `where` command uses the same expression syntax as [eval](#). Also, both commands interpret quoted strings as literals. If the string is not quoted, it is treated as a field. Because of this, you can use `where` to compare two different fields, which you cannot use `search` to do.

Examples

Example 1: 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)
```

Example 2: 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

The `x11` exposes the seasonal pattern in your time-based data series so that you can subtract it from the underlying data and see the real trend. This command has a similar purpose to the [trendline command](#), but it uses the more sophisticated and industry popular X11 method.

For more information, read "About predictive analytics with Splunk" in the Search Manual.

Synopsis

Remove seasonal fluctuations in fields.

Syntax

```
x11 [<type>] [<period>=<int>] (<fieldname>) [as <newname>]
```

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. Defaults to mult().

<period>

Syntax: <int>

Description: The period of the data relative to the number of data points, expressed as an integer between 5 and 10000. If the period is 7, the command expects the data to be periodic ever 7 data points. If not supplied, Splunk computes 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 10000.

<newname>

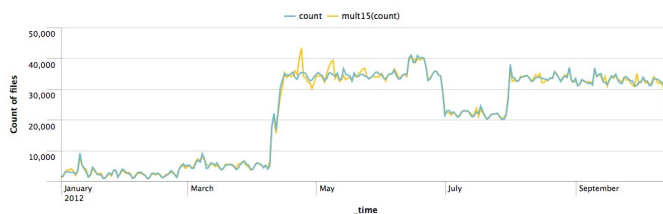
Syntax: <string>

Description: Specify a field name for the output of x11. Otherwise, defaults to the specified "<type><period>(<fieldname>)".

Examples

Example 1: Here type is the default 'mult' and period is 15.

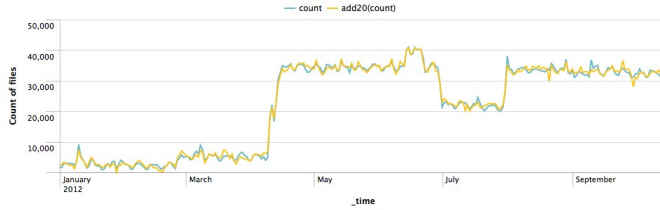
```
index=download | timechart span=1d count(file) as count | x11  
mult15(count)
```



Note: Here, because the span=1d, every data point accounts for 1 day. And, as a result, the period in this example is 15 days.

Example 2: Here type is 'add' and period is 20.

```
iindex=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

Synopsis

Extracts `xml` key-value pairs.

Syntax

```
xmlkv maxinputs=<int>
```

Required arguments

maxinputs

Syntax: maxinputs=<int>

Description:

Description

Finds key value pairs of the form `<foo>bar</foo>` where `foo` is the key and `bar` is the value from the `_raw` key.

Examples

Example 1: Extract field/value pairs from XML formatted data. "xmlkv" automatically extracts values between XML tags.


```
... | xmlkv
```

Example 2: Example usage

```
... | xmlkv maxinputs=10000
```

See also

[extract](#), [kvform](#), [multikv](#), [rex](#), [xpath](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `xmlkv` command.

xmlunescape

Synopsis

Un-escapes `xml` characters.

Syntax

`xmlunescape maxinputs=<int>`

Required arguments

`maxinputs`

Syntax: `maxinputs=<int>`

Description:

Description

Un-escapes `xml` entity references (for: `&`, `<`, and `>`) back to their corresponding characters (e.g., `&` -> `&`).

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

Synopsis

Extracts the `xpath` value from `field` and sets the `outfield` attribute.

Syntax

```
xpath [outfield=<field>] <string:xpath> [field=<field>] [default=<string>]
```

Required arguments

`xpath`

Syntax: `<string>`

Description: Specify the XPath reference.

Optional arguments

`field`

Syntax: `field=<field>`

Description: The field to find and extract the referenced `xpath` value.
Defaults to `_raw`.

`outfield`

Syntax: `outfield=<field>`

Description: The field to write the `xpath` value. Defaults to `xpath`.

`default`

Syntax: `default=<string>`

Description: If the attribute referenced in `xpath` doesn't exist, this specifies what to write to `outfield`. If this isn't defined, there is no default value.

Description

Sets the value of `outfield` to the value of the `xpath` applied to `field`.

Examples

Example 1: Extract the `name` value from `_raw` XML events, which might look like this:

```
<foo>
<bar name="spock">
</bar>
</foo>
```

```
sourcetype="xml" | xpath outfield=name "//bar/@name"
```

Example 2: Extract the `identity_id` and `instrument_id` from the `_raw` XML events:

```
<DataSet xmlns="">
  <identity_id>3017669</identity_id>
  <instrument_id>912383KM1</instrument_id>
  <transaction_code>SEL</transaction_code>
  <sname>BARC</sname>
  <currency_code>USA</currency_code>
</DataSet>

<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"
```

This search will return two results: `identity_id=3017669` and `identity_id=1037669`.

```
... | xpath outfield=instrument_id
      "//DataSet[sname=\"BARC\"]/instrument_id"
```

Because you specify `sname="BARC"`, this search will return one result: `instrument_id=912383KM1`.

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.

xyseries

Synopsis

Converts results into a format suitable for graphing.

Syntax

```
xyseries [grouped=<bool>] <x-field> <y-name-field> <y-data-field>...  
[sep=<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

grouped

Syntax: grouped= true | false

Description: If true, indicates that the input is sorted by the value of the <x-field> and multi-file input is allowed. Defaults to false.

sep

Syntax: sep=<string>

Description:

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.

Internal Search Commands

About internal commands

Internal search commands refer to search commands that are experimental. They may be removed or updated and reimplemented differently in future versions. They are not supported commands.

collapse

The collapse command is an experimental command and not supported by Splunk.

Synopsis

Condenses multi-file results into as few files as chunksize option will allow.

Syntax

... | collapse [chunksize=<num>] [force=<bool>]

Optional arguments

chunksize

Syntax: chunksize=<num>

Description: Limits the number of resulting files. Default is 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 is false.

Description

The collapse command is automatically invoked by output* operators.

Examples

Example 1: Collapse results.

```
... | collapse
```

dispatch

The dispatch command is no longer required; all Splunk searches are run as dispatch searches. For more information, see [the search command](#).

runshellscript

The runshellscript command is an internal command used to execute scripted alerts. Currently, it is not supported by Splunk.

Synopsis

Execute scripted alerts.

Syntax

```
runshellscript <script-filename> <result-count> <search-terms> <search-string>  
<savedsearch-name> <description> <results-url> <deprecated-arg> <search-id>  
<results_file>
```

Description

Internal command used to execute scripted alerts. The script file needs to be located in either `$(SPLUNK_HOME)/etc/system/bin/scripts` OR `$(SPLUNK_HOME)/etc/apps/<app-name>/bin/scripts`. The search ID is used to create a path to the search's results. All other arguments are passed to the script (unvalidated) as follows:

Argument	Description
\$0	The filename of the script.
\$1	The result count, or number of events returned.
\$2	The search terms.
\$3	The fully qualified query string.

\$4	The name of the saved search in Splunk.
\$5	The description or trigger reason (i.e. "The number of events was greater than 1").
\$6	The link to saved search results.
\$7	DEPRECATED - empty string argument.
\$8	The search ID
\$9	The path to the results file, results.csv. (Contains raw results.)

For more information, check out this excellent topic on troubleshooting alert scripts on the Splunk Community Wiki and see "Configure scripted alerts" in the Alerting Manual.

See also

[script](#)

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the runshellsript command.

tscollect

The `tscollect` command is an internal command used to save search results into a `tsidx` formatted file. Currently, it is an experimental command and not supported by Splunk.

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've 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.

`tscollect` can create multiple `tsidx` files in the same namespace. It will begin a new `tsidx` file when it determines that the one it's currently creating has gotten big enough.

Synopsis

Writes results into tsidx file(s) for later use by [tstats command](#).

Important: The 'indexes_edit' capability is required to run this command.

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). Defaults to 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. 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. This namespace location is also configurable in `index.conf`, with 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 similar to Splunk's, set squashcase=true for results to be converted to all lowercase. Defaults to false.

Examples

Example 1: Write the results table to tsidx files in namespace foo.

```
... | tscollect namespace=foo
```

Example 2: Write the values of field `foo` for the events in the main index to `tsidx` files in the job directory.

```
index=main | fields foo | tscollect
```

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

The `tstats` command is an internal command used to calculate statistics over `tsidx` files created with the `tscollect` command. Currently, it is an experimental command and not supported by Splunk.

When you want to report on very large data sets, use the `tscollect` command to save search results into a **tsidx file** that exists in a specific namespace (that you create with the `tscollect` command).

Then use the `tstats` command to calculate statistics on the data summarized into the `tsidx` file. Because you are not reading events from raw data, you can expect significantly faster search and reporting performance. `tstats` operates in a manner similar to that of `stats`; the primary differences are that:

- it is a generating processor, so it must be the first command in a search
- it uses a smaller set of stats functions
- it requires you to specify the namespace for the target `tsidx` file or the job id of the `tscollect` job

Since `tstats` does not support all the functionality of the normal `stats` command, you have the option to output results in the `prestats` format for use by `stats`, which combines the speed of `tstats` with all the functionality of `stats`. Operating in `prestats` mode also enables preview for results, so this is highly recommended for large data sets.

Note: Except in `prestats` and `append` modes (`prestats=t` and `append=t`), this command is a generating processor, so it must be the first command in a search.

See the Syntax below for more details.

Synopsis

Performs statistical queries on tsidx files created using [tscollect](#).

Syntax

```
| tstats [append=<bool>] [prestats=<bool>] <aggregate-opt>... FROM  
<namespace|tscollect-job-id> [WHERE <search_query>] [GROUPBY <field-list>  
[span=<timespan>] ]
```

Required arguments

aggregate-opt

Syntax:

count|count(<field>)|sum(<field>)|sumsq(<field>)|distinct(<field>)|avg(<field>)|stdev(<field>)
[AS <string>]

Description: Either perform a basic count, get the values of a field, or perform a function. You can also rename the result using 'AS'. While there are only a few directly supported functions in tstats, if you are running with the prestats option (and only then) you can supply any function that stats supports with <stats-fn>.

namespace

Syntax: <string>

Description: Define a location for the tsidx file with

\$SPLUNK_DB/tsidxstats. This namespace location is also configurable in `index.conf`, with the attribute `tsidxStatsHomePath`.

tscollect-job-id

Syntax: <string>

Description: The job ID of a tscollect search.

Optional arguments

append

Syntax: append=<bool>

Description: When in prestats mode (`prestats=t`), enables `append=t` where the prestats results append to any input results.

prestats

Syntax: prestats=<bool>

Description: Use this to perform any [stats](#) function that `tstats` does not support (is not listed as an aggregate option). When true, this option also enables preview for results. For more information see [Functions for stats, chart, and timechart](#). Defaults to false.

<field-list>

Syntax: <field>, <field>, ...

Description: Specify a list of fields to group results.

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.

Grouping by _time

You can provide any number of `GROUPBY` fields. If you are grouping by `_time`, you should supply a timespan for grouping the time buckets. This timespan looks like any normal timespan in Splunk, `span='1hr'` or `'3d'`.

Examples

Example 1: Gets the count of all events in the `mydata` namespace.

```
| tstats count FROM mydata
```

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 split by each day for all the data in `mydata`

```
| tstats count from mydata GROUPBY _time span=1d
```

Example 4: Uses `prestats` mode to calculate the median of the field `foo`.

```
| tstats prestats=t median(foo) FROM mydata | stats median(foo)
```

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 my otherdata | stats median(foo) median(bar)
```

See also

[stats](#), [tscollect](#)

Answers

Have questions? Visit [Splunk Answers](#) and see what questions and answers the Splunk community has using the `tstats` command.

Search in the CLI

About searches in the CLI

You can use the Splunk **CLI** to monitor, configure, and execute searches on your Splunk server. 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 both historical and real-time searches from the CLI in Splunk by invoking the `search` or `rtsearch` commands, respectively. 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>
```

Object	Description
rtsearch	Returns the parameters and syntax for real-time searches.
search	Returns the parameters and syntax for historical searches.
search-commands	Returns a list of search commands that you can use from the CLI.
search-fields	Returns a list of default fields.
search-modifiers	Returns a list of search and time-based modifiers that you can use to narrow your search.

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 searches you run from Splunk Web except that you can pass parameters outside of the query to control

the time limit of the search, tell Splunk which server to run the search, and specify how Splunk displays results.

- 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, tell Splunk which server to run the search, and specify how Splunk displays results.

```
search | rtsearch [object][-parameter <value>]
```

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 in Splunk, see the "Start searching" topic in the *Splunk Tutorial*.
- For the complete list of search commands, see ["All search commands"](#) in the Search Reference Manual.
- For a quick reference search language and search commands, see the ["Search Command Cheat Sheet and Search Language Quick Reference Card"](#) in the Search Reference Manual.

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 Splunk 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.

Parameter	Value(s)	Default(s)	Description
app	<app_name>	search	Specify the name of the app in which to run your search.
batch	<bool>	F	Indicates how to handle updates in preview mode.
detach	<bool>	F	Triggers an asynchronous search and displays the job ID and TTL for the search.
earliest_time	<time-modifier>	–	The relative time modifier for the start time of the search. This is optional for both search and rtsearch.
header	<bool>	T	Indicates whether to display a header in the table output mode.
latest_time	<time-modifier>	–	The relative time modifier for the end time of search. For search, if this is not specified, it defaults to the end of the time (or the time of the last event in the data),

			so that any "future" events are also included. For rtsearch, this is a required parameter and the real-time search will not run if it's not specified.
max_time	<number>	0	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.
maxout	<number>	search, 100 rtsearch, 0	The maximum number of events to return or send to stdout (when exporting events). The maximum allowable value is 10000. A value of 0 means that it will output an unlimited number of events.
output	rawdata, table, csv, auto	For non-transforming searches, rawdata. For transforming searches, table.	Indicates how to display the job.
preview	<bool>	T	Indicates that reporting searches should be previewed (displayed as results are calculated).

timeout	<number>	0	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.
uri	[http https]://name_of_server:management_port		<p>Specify the server name and management port. <code>name_of_server</code> can be the fully-resolved domain name or the IP address of the Splunk server.</p> <p>The default uri value is the <code>mgmtHostPort</code> value that you defined in the Splunk server's <code>web.conf</code>.</p> <p>For more information, see Access and use the CLI on a remote Splunk Server in the <i>Admin manual</i>.</p>
wrap	<bool>	T	Indicates whether to line wrap for individual lines that are longer than the terminal width.

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.

```
./splunk search 'eventtype=webaccess error' -detach true
```

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.