

SEARCH LANGUAGE

A *search* is a series of commands and arguments, each chained together with "|" (pipe) character that takes the output of one command and feeds it into the next command on the right.

```
search-args | cmd1 cmd-args | cmd2 cmd-args | ...
```

Search commands are used to take indexed data and filter unwanted information, extract more information, calculate values, transform, and statistically analyze. The search results retrieved from the index can be thought of as a dynamically created table. Each search command redefines the shape of that table. Each indexed event is a row, with columns for each field value. Columns include basic information about the data as well as columns that are dynamically extracted at search-time.

At the head of each search is an implied search-the-index-for-events command, which can be used to search for keywords (e.g., `error`), boolean expressions (e.g., `(error OR failure) NOT success`), phrases (e.g., `"database error"`), wildcards (e.g., `fail*` will match fail, fails, failure, etc.), field values (e.g., `code=404`), inequality (e.g., `code!=404` or `code>200`), a field having any value or no value (e.g., `code=*` or `NOT code=*`). For example, the search:

```
sourcetype="access_combined" error | top 10 uri
```

will retrieve indexed `access_combined` events from disk that contain the term `"error"` (ANDs are implied between search terms), and then for those events, report the top 10 most common URI values.

Subsearches

A *subsearch* is an argument to a command that runs its own search, returning those results to the parent command as the argument value. *Subsearches* are contained in square brackets. For example, finding all syslog events from the user that had the last login error:

```
sourcetype=syslog [ search login error | return 1 user ]
```

Relative Time Modifiers

Besides using the custom-time ranges in the user-interface, you can specify in your search the time ranges of retrieved events with the `latest` and `earliest` search modifiers. The relative times are specified with a string of characters that indicate amount of time (integer and unit) and, optionally, a "snap to" time unit:

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

For example: `"error earliest=-1d@d latest=-1h@h"` will retrieve events containing `"error"` that occurred from yesterday (snapped to midnight) to the last hour (snapped to the hour).

Time Units: specified as second (s), minute(m), hour(h), day(d), week(w), month(mon), quarter(q), year(y). "time_integer" defaults to 1 (e.g., "m" is the same as "1m").

Snapping: indicates the nearest or latest time to which your time amount rounds down. Snaps rounds down to the latest time not after the specified time. For example, if it is 11:59:00 and you "snap to" hours (@h), you will snap to 11:00 not 12:00. You can "snap to" a specific day of the week: use @w0 for Sunday, @w1 for Monday, etc.

COMMON SEARCH COMMANDS

COMMAND	DESCRIPTION
chart/timechart	Returns results in a tabular output for (time-series) charting.
dedup	Removes subsequent results that match a specified criterion.
eval	Calculates an expression. (See EVAL FUNCTIONS table.)
fields	Removes fields from search results.
head/tail	Returns the first/last N results.
lookup	Adds field values from an external source.
rename	Renames a specified field; wildcards can be used to specify multiple fields.
replace	Replaces values of specified fields with a specified new value.
rex	Specifies regular expression named groups to extract fields.
search	Filters results to those that match the search expression.
sort	Sorts search results by the specified fields.
stats	Provides statistics, grouped optionally by fields.
top/rare	Displays the most/least common values of a field.
transaction	Groups search results into transactions.

Optimizing Searches

The key to fast searching is to limit the data that needs to be pulled off disk to an absolute minimum, and then to filter that data as early as possible in the search so that processing is done on the minimum data necessary.

Partition data into separate indexes, if you'll rarely perform searches across multiple types of data. For example, put web data in one index, and firewall data in another.

- Search as specifically as you can (e.g. `fatal_error`, not `*error*`)
- Limit the time range to only what's needed (e.g., -1h not -1w)
- Filter out unneeded fields as soon as possible in the search.
- Filter out results as soon as possible before calculations.
- For report generating searches, use the Advanced Charting view, and not the Flashtimeline view, which calculates timelines.
- On Flashtimeline, turn off 'Discover Fields' when not needed.
- Use summary indexes to pre-calculate commonly used values.
- Make sure your disk I/O is the fastest you have available.

EVAL FUNCTIONS

The *eval* command calculates an expression and puts the resulting value into a field (e.g. `...| eval force = mass * acceleration`). The following table lists the functions eval understands, in addition to basic arithmetic operators (+ * / %), string concatenation (e.g., `...| eval name = last . " . last`), boolean operations (AND OR NOT XOR <> <= >= != == LIKE).

FUNCTION	DESCRIPTION	EXAMPLES
abs (X)	Returns the absolute value of X.	<code>abs(number)</code>
case (X, "Y" , ...)	Takes pairs of arguments X and Y, where X arguments are Boolean expressions that, when evaluated to TRUE, return the corresponding Y argument.	<code>case(error == 404, "Not found", error == 500, "Internal Server Error", error == 200, "OK")</code>
ceil (X)	Ceiling of a number X.	<code>ceil(1.9)</code>
cidrmatch ("X" , Y)	Identifies IP addresses that belong to a particular subnet.	<code>cidrmatch("123.132.32.0/25", ip)</code>
coalesce (X, ...)	Returns the first value that is not null.	<code>coalesce(null(), "Returned val", null())</code>
exact (X)	Evaluates an expression X using double precision floating point arithmetic.	<code>exact(3.14*num)</code>
exp (X)	Returns e ^X .	<code>exp(3)</code>
floor (X)	Returns the floor of a number X.	<code>floor(1.9)</code>
if (X, Y , Z)	If X evaluates to TRUE, the result is the second argument Y. If X evaluates to FALSE, the result evaluates to the third argument Z.	<code>if(error==200, "OK", "Error")</code>
isbool (X)	Returns TRUE if X is Boolean.	<code>isbool(field)</code>
isint (X)	Returns TRUE if X is an integer.	<code>isint(field)</code>
isnotnull (X)	Returns TRUE if X is not NULL.	<code>isnotnull(field)</code>
isnull (X)	Returns TRUE if X is NULL.	<code>isnull(field)</code>
isnum (X)	Returns TRUE if X is a number.	<code>isnum(field)</code>
isstr ()	Returns TRUE if X is a string.	<code>isstr(field)</code>
len (X)	This function returns the character length of a string X.	<code>len(field)</code>
like (X, "Y")	Returns TRUE if and only if X is like the SQLite pattern in Y.	<code>like(field, "foo%")</code>
ln (X)	Returns its natural log.	<code>ln(bytes)</code>
log (X, Y)	Returns the log of the first argument X using the second argument Y as the base. Y defaults to 10.	<code>log(number, 2)</code>
lower (X)	Returns the lowercase of X.	<code>lower(username)</code>
ltrim (X, Y)	Returns X with the characters in Y trimmed from the left side. Y defaults to spaces and tabs.	<code>ltrim(" ZZZabcZZ ", " Z")</code>
match (X, Y)	Returns if X matches the regex pattern Y.	<code>match(field, "^\\d{1,3}\\.\d\$")</code>
max (X, ...)	Returns the max.	<code>max(delay, mydelay)</code>
md5 (X)	Returns the MD5 hash of a string value X.	<code>md5(field)</code>
min (X, ...)	Returns the min.	<code>min(delay, mydelay)</code>
mvcount (X)	Returns the number of values of X.	<code>mvcount(multifield)</code>
mvfilter (X)	Filters a multi-valued field based on the Boolean expression X.	<code>mvfilter(match(email, "net\$"))</code>
mvindex (X, Y , Z)	Returns a subset of the multivalued field X from start position (zero-based) Y to Z (optional).	<code>mvindex(multifield, 2)</code>
mvjoin (X, Y)	Given a multi-valued field X and string delimiter Y, and joins the individual values of X using Y.	<code>mvjoin(foo, ";")</code>
now ()	Returns the current time, represented in Unix time.	<code>now()</code>
null ()	This function takes no arguments and returns NULL.	<code>null()</code>
nullif (X, Y)	Given two arguments, fields X and Y, and returns the X if the arguments are different; returns NULL, otherwise.	<code>nullif(fieldA, fieldB)</code>
pi ()	Returns the constant pi.	<code>pi()</code>
pow (X, Y)	Returns X ^Y .	<code>pow(2, 10)</code>
random ()	Returns a pseudo-random number ranging from 0 to 2147483647.	<code>random()</code>
relative_time (X, Y)	Given epochtime time X and relative time specifier Y, returns the epochtime value of Y applied to X.	<code>relative_time(now(), "-1d@d")</code>
replace (X, Y, Z)	Returns a string formed by substituting string Z for every occurrence of regex string Y in string X.	Returns date with the month and day numbers switched, so if the input was 1/12/2009 the return value would be 12/1/2009: <code>replace(date, "^\\d{1,2}/\\d{1,2}/", "\\2/\\1/")</code>
round (X, Y)	Returns X rounded to the amount of decimal places specified by Y. The default is to round to an integer.	<code>round(3.5)</code>
rtrim (X, Y)	Returns X with the characters in Y trimmed from the right side. If Y is not specified, spaces and tabs are trimmed.	<code>rtrim(" ZZZabcZZ ", " Z")</code>

EVAL FUNCTIONS (continued)

FUNCTION	DESCRIPTION	EXAMPLES
searchmatch (X)	Returns true if the event matches the search string X.	<code>searchmatch("foo AND bar")</code>
split (X, "Y")	Returns X as a multi-valued field, split be delimiter Y.	<code>split(foo, ";")</code>
sqrt (X)	Returns the square root of X.	<code>sqrt(9)</code>
strftime (X, Y)	Returns epochtime value X rendered using the format specified by Y.	<code>strftime(_time, "%H:%M")</code>
strptime (X, Y)	Given a time represented by a string X, returns value parsed from format Y.	<code>strptime(timeStr, "%H:%M")</code>
substr (X, Y, Z)	Returns a substring field X from start position (1-based) Y for Z (optional) characters.	<code>substr("string", 1, 3)</code> <code>+substr("string", -3)</code>
time ()	Returns the wall-clock time with microsecond resolution.	<code>time()</code>
tonumber (X, Y)	Converts input string X to a number, where Y (optional, defaults to 10) defines the base of the number to convert to.	<code>tonumber("0A4", 16)</code>
tostring (X, Y)	Returns a field value of X as a string. If the value of X is a number, it reformats it as a string; if a Boolean value, either "True" or "False". If X is a number, the second argument Y is optional and can either be "hex" (convert X to hexadecimal), "commas" (formats X with commas and 2 decimal places), or "duration" (converts seconds X to readable time format HH:MM:SS).	This example returns: <code>foo=615</code> and <code>foo2=00:10:15</code> : <code>... eval foo=615 eval foo2 = tostring(foo, "duration")</code>
trim (X, Y)	Returns X with the characters in Y trimmed from both sides. If Y is not specified, spaces and tabs are trimmed.	<code>trim(" ZZZZabcZZ ", " Z")</code>
typeof (X)	Returns a string representation of its type.	This example returns: <code>"NumberStringBoolInvalid"</code> : <code>typeof(12)+ typeof("string")+ typeof(1==2)+ typeof(badfield)</code>
upper (X)	Returns the uppercase of X.	<code>upper(username)</code>
urldecode (X)	Returns the URL X decoded.	<code>urldecode("http%3A%2F%2Fwww.splunk.com%2Fdownload%3Fr%3Dheader")</code>
validate (X, Y, ...)	Given pairs of arguments, Boolean expressions X and strings Y, returns the string Y corresponding to the first expression X that evaluates to False and defaults to NULL if all are True.	<code>validate(isint(port), "ERROR: Port is not an integer", port >= 1 AND port <= 65535, "ERROR: Port is out of range")</code>

COMMON STATS FUNCTIONS

Common statistical functions used with the `chart`, `stats`, and `timechart` commands. Field names can be wildcarded, so `avg(*delay)` might calculate the average of the `delay` and `xdelay` fields.

FUNCTION	DESCRIPTION
avg (X)	Returns the average of the values of field X.
count (X)	Returns the number of occurrences of the field X. To indicate a specific field value to match, format X as <code>eval(field="value")</code> .
dc (X)	Returns the count of distinct values of the field X.
first (X)	Returns the first seen value of the field X. In general, the first seen value of the field is the chronologically most recent instance of field.
last (X)	Returns the last seen value of the field X.
list (X)	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.
max (X)	Returns the maximum value of the field X. If the values of X are non-numeric, the max is found from lexicographic ordering.
median (X)	Returns the middle-most value of the field X.
min (X)	Returns the minimum value of the field X. If the values of X are non-numeric, the min is found from lexicographic ordering.
mode (X)	Returns the most frequent value of the field X.
perc<X> (Y)	Returns the X-th percentile value of the field Y. For example, <code>perc5(total)</code> returns the 5th percentile value of a field "total".
range (X)	Returns the difference between the max and min values of the field X.
stdev (X)	Returns the sample standard deviation of the field X.
stdevp (X)	Returns the population standard deviation of the field X.
sum (X)	Returns the sum of the values of the field X.
sumsq (X)	Returns the sum of the squares of the values of the field X.
values (X)	Returns the list of all distinct values of the field X as a multi-value entry. The order of the values is lexicographical.
var (X)	Returns the sample variance of the field X.

splunk™ Community

ask questions, find answers.
download apps, share yours.

community.splunk.com