**eval**

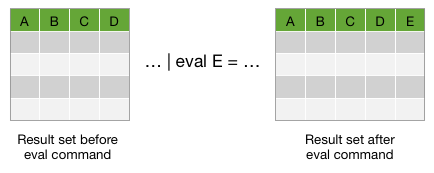
**Description**

The eval command calculates an expression and puts the resulting value into a destination field. If this destination field matches a field name that already exists, it overwrites the existing field value with the results of the eval expression. The eval command evaluates mathematical, string, and boolean expressions.

You can chain multiple eval expressions in one search using a comma to separate subsequent expressions. The search processes multiple eval expressions left-to-right and lets you reference previously evaluated fields in subsequent expressions.

**Difference between eval and stats commands**

The stats command calculates statistics based on fields in your events. The eval command creates new fields in your events by using existing fields and an arbitrary expression.

[](http://docs.splunk.com/File:Eval1.png)

**Syntax**

eval <field>=<expression>["," <field>=<expression>]...

**Required arguments**

**field**

**Syntax:** <string>

**Description:** A destination field name for the resulting calculated value. If the field name already exists in your events, eval overwrites the value.

**expression**

**Syntax:** <string>

**Description:** A combination of values, variables, operators, and functions that will be executed to determine the value to place in your destination field.

The syntax of the eval expression is checked before running the search, and an exception is thrown for an invalid expression.

* The result of an eval statement is not allowed to be boolean. If, at search time, the expression cannot be evaluated successfully for a given event, eval erases the resulting field.
* If the expression references a **field name** that contains non-alphanumeric characters, it needs to be surrounded by**single quotation marks**. For example, if the field name is server-1 you specify the field name like thisnew=count+'server-1'.
* If the expression references **literal strings** it needs to be surrounded by **double quotation marks**. For example, if the string you want to use is server- you specify the string like this new="server-"+host.

**Operators**

The following table lists the basic operations you can perform with the eval command. For these evaluations to work, the values need to be valid for the type of operation. For example, with the exception of addition, arithmetic operations might not produce valid results if the values are not numerical. When concatenating values, Splunk software reads the values as strings, regardless of the value.

|  |  |
| --- | --- |
| **Type** | **Operators** |
| Arithmetic | + - \* / % |
| Concatenation | . |
| Boolean | AND OR NOT XOR < > <= >= != = == LIKE |

Operators that produce numbers

* The plus ( + ) operator accepts two numbers for addition, or two strings for concatenation.
* The subtraction ( - ), multiplication ( \* ), division ( / ), and modulus ( % ) operators accept two numbers.

Operators that produce strings

* The period ( . ) operator concatenates both strings and number. Numbers are concatenated in their string represented form.

Operators that produce booleans

* The AND, OR, NOT, and XOR operators accept two Boolean values.
* The <>, <=, !=, and == operators accept two numbers or two strings. </code>, !=, and == operators accept two numbers or two strings. The single equal sign ( = ) is a synonym for the double equal sign ( == ).
* The LIKE operator accepts two strings. This is a pattern match similar to what is used in SQL. For example string LIKE pattern. The pattern operator supports literal text, a percent ( % ) character for a wildcard, and an underscore ( \_ ) character for a single character match. For example, field LIKE "a%b\_" matches any string starting with a, followed by anything, followed by b, followed by one character.

**Functions**

You can use the following functions with the eval command: 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, mvrange, mvzip, now, null, nullif, pi, pow, random, relative\_time, replace, round, rtrim, searchmatch, sha1, sha256, sha512, sigfig, spath, split, sqrt, strftime, strptime, substr, time, tonumber, tostring, trim, typeof, upper, urldecode, validate.

For **descriptions and examples** of each function, see ["Evaluation functions"](http://docs.splunk.com/Documentation/Splunk/6.4.1/SearchReference/CommonEvalFunctions).

**Usage**

**General**

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.

You can also use the value of another field as the name of the destination field by using curly brackets, { }. For example, if you have an event with the following fields, aName=counter and field aValue=1234, use | eval {aName}=aValue to return counter=1234.

Numbers and strings can be assigned to fields, while booleans may not be. However you can convert booleans and nulls to strings using tostring(), which may be assigned to fields.

During calculations, numbers are double precision floating point numbers subject to all the usual behaviors of floating point numbers. Operations resulting in NaN assigned to a field will result in "nan". Positive and negative overflow will result in "inf" and "-inf". Division by zero will result in a null field.

If you are using a search as an argument to the eval command and functions, you cannot use a saved search name; you must pass a literal search string or a field that contains a literal search string (like the 'search' field extracted from index=\_audit events).

**Calculated fields**

You can use eval statements to define calculated fields by defining the eval statement in props.conf. When you run a search, Splunk software evaluates the statements and creates fields in a manner similar to that of search time field extraction. Setting up calculated fields means that you no longer need to define the eval statement in a search string. Instead, you can search on the resulting calculated field directly.

You can use calculated fields to move your commonly used eval statements out of your search string and intoprops.conf, where they will be processed behind the scenes at search time. With calculated fields, you can change the search from:

sourcetype="cisco\_esa" mailfrom=\* | eval accountname=split(mailfrom,"@"), from\_user=mvindex(accountname,0), from\_domain=mvindex(accountname,-1) | table mailfrom, from\_user, from\_domain

to this search:

sourcetype="cisco\_esa" mailfrom=\* | table mailfrom, from\_user, from\_domain

In this example, the three eval statements that were in the search--that defined the accountname, from\_user, andfrom\_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](http://docs.splunk.com/Documentation/Splunk/6.4.1/Knowledge/definecalcfields) in the *Knowledge Manager Manual*.

**Search event tokens**

If you are using the eval command in search event tokens, some of the evaluation functions might be unavailable or have a different behavior. See ["Custom logic for search tokens"](http://docs.splunk.com/Documentation/Splunk/6.4.1/Viz/tokens#Custom_logic_for_search_tokens) in *Dashboards and Visualizations* for information about the evaluation functions that you can use with search event tokens.

**Basic Examples**

**1. Create a new field that contains the result of a calculation**

Create a new field called velocity in each event. Calculate the velocity by dividing the values in the distance field by the values in the time field.

... | eval velocity=distance/time

**2. Use the if function to determine the values placed in the status field**

Create a field called status in each event. Set the value in the status field to OK if the error value is 200. Otherwise set the status value to Error.

... | eval status = if(error == 200, "OK", "Error")

**3. Convert values to lowercase**

Create a new field in each event called lowuser and populate the field with the lowercase version of the values in theusername field.

... | eval lowuser = lower(username)

**4. 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)

**5. Set status to some simple http error codes**

... | eval error\_msg = case(error == 404, "Not found", error == 500, "Internal Server Error", error == 200, "OK")

**6. Set full\_name to the concatenation of first\_name, a space, and last\_name**

... | eval full\_name = first\_name." ".last\_name

**7. Separate multiple eval commands with a comma**

... | eval full\_name = first\_name." ".last\_name, low\_name = lower(full\_name)

**8. Display timechart of the avg of cpu\_seconds by processor, rounded to 2 decimals**

... | timechart eval(round(avg(cpu\_seconds),2)) by processor

**9. Convert a numeric field value to a string with commas and 2 decimals**

If the original value of x is 1000000, this returns x as 1,000,000.

... | eval x=tostring(x,"commas")

To include a currency symbol at the beginning of the string:

... | eval x="$".tostring(x,"commas")

This returns x as $1,000,000.

**Extended Examples**

**10. Coalesce a field from two different source types, create a transaction of events**

This example shows how you might coalesce a field from two different source types and use that to create a transaction of events. sourcetype=A has a field called number, and sourcetype=B has the same information in a field called subscriberNumber.

sourcetype=A OR sourcetype=B | eval phone=coalesce(number,subscriberNumber) | transaction phone maxspan=2m

The eval command is used to add a common field, called phone, to each of the events whether they are fromsourcetype=A or sourcetype=B. The value of phone is defined, using the coalesce() function, as the values of numberand 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.

**11. Separate events into categories, count and display minimum and maximum values**

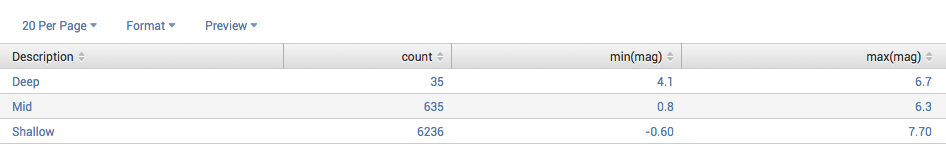
|  |
| --- |
| This example uses recent earthquake data downloaded from the [USGS Earthquakes website](http://earthquake.usgs.gov/earthquakes/). The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), etc., for each earthquake recorded.  You can download a current CSV file from the [**USGS Earthquake Feeds**](http://earthquake.usgs.gov/earthquakes/feed/v1.0/csv.php) and add it as an input.  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=usgs | eval Description=case(depth<=70, "Shallow", depth>70 AND depth<=300, "Mid", depth>300, "Deep") | stats count min(mag) max(mag) by Description

The eval command is used to create a field called Description, which takes the value of "Shallow", "Mid", or "Deep" based on the Depth of the earthquake. The case() function is used to specify which ranges of the depth fits each description. For example, if the depth is less than 70 km, the earthquake is characterized as a shallow-focus quake; and the resulting Description is Shallow.

The search also pipes the results of eval into the stats command to count the number of earthquakes and display the minimum and maximum magnitudes for each Description:

[](http://docs.splunk.com/File:Searchref_eval_usgsex1.1.png)

**12. Find IP addresses and categorize by network using eval functions cidrmatch and if**

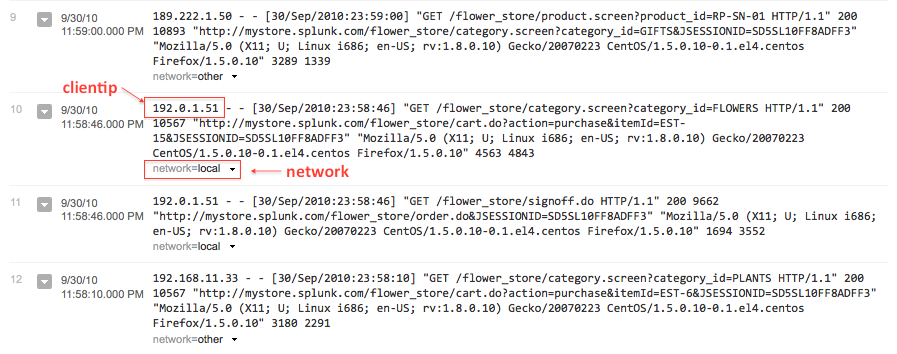
|  |
| --- |
| This example is designed to use the sample dataset from [**"Get the tutorial data into Splunk"**](http://docs.splunk.com/Documentation/Splunk/6.4.1/SearchTutorial/GetthetutorialdataintoSplunk) topic of the Search Tutorial, but it should work with any format of Apache Web access log. Download the data set and follow the instructions in that topic to upload it to Splunk Enterprise. 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("192.168.0.0/16", 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:

[](http://docs.splunk.com/File:EvalEx2_events.png)

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 13)

**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**](http://docs.splunk.com/Documentation/Splunk/6.4.1/Knowledge/Abouteventtypes) in the *Knowledge manager manual*.

**13. Extract information from an event into a separate field, create a multivalue field**

|  |
| --- |
| 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). |

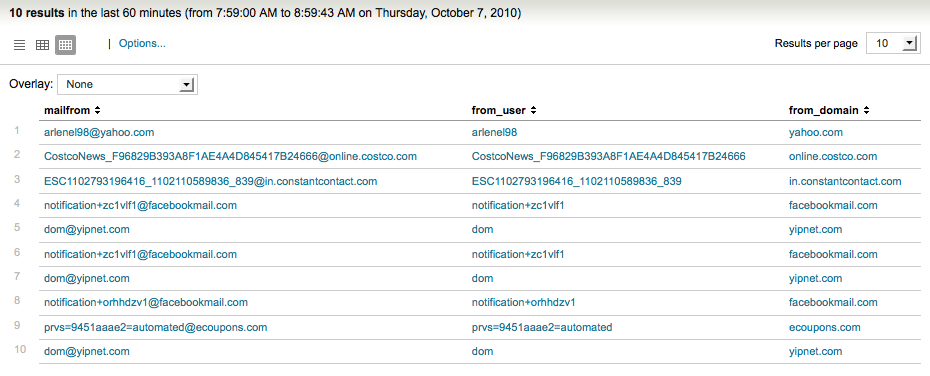
Use the email address field to extract the user's name and domain. The eval command in this search contains multiple expressions, separated by commas.

sourcetype="cisco\_esa" mailfrom=\* | eval accountname=split(mailfrom,"@"), from\_user=mvindex(accountname,0), from\_domain=mvindex(accountname,-1) | table mailfrom, from\_user, from\_domain

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 ofaccountname, respectively.

The results of the eval expressions are then piped into the table command. You can see the the original mailfromvalues and the new from\_user and from\_domain values in the following results table:

[](http://docs.splunk.com/File:EvalEx4_results.png)

**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).

**14. Categorize events using the match function**

|  |
| --- |
| 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 are not necessarily from *abroad*.)

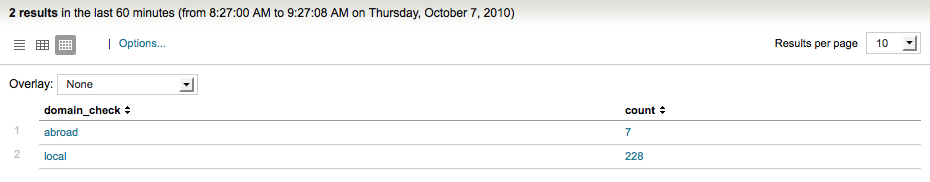
The eval command in this search contains multiple expressions, separated by commas.

sourcetype="cisco\_esa" mailfrom=\*| eval accountname=split(mailfrom,"@"), from\_domain=mvindex(accountname,-1), location=if(match(from\_domain, "[^\n\r\s]+\.(com|net|org)"), "local", "abroad") | stats count by location

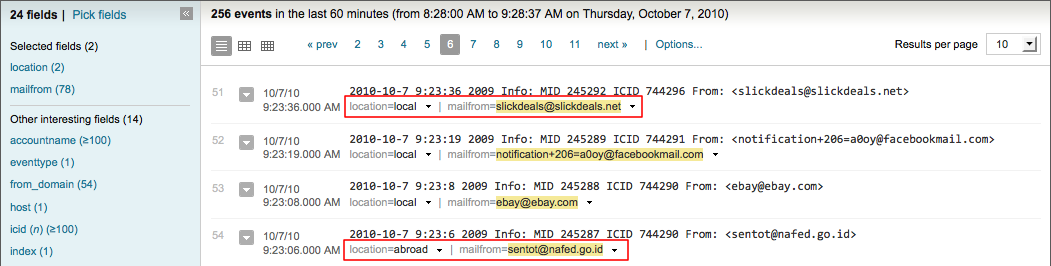
The first half of this search is similar to Example 12. The split() function is used to break up the email address in themailfrom field. The mvindex function defines the from\_domain as the portion of the mailfrom field after the @ symbol.

Then, the if() and match() functions are used: if the from\_domain value ends with a .com, .net., or .org, thelocation 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:

[](http://docs.splunk.com/File:EvalEx5_resultsTable.png)

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:

[](http://docs.splunk.com/File:EvalEx5_eventsList.png)

**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**](http://docs.splunk.com/Documentation/Splunk/6.4.1/Knowledge/Abouteventtypes) in the *Knowledge manager manual*.

**15. Convert the duration of transactions into more readable string formats**

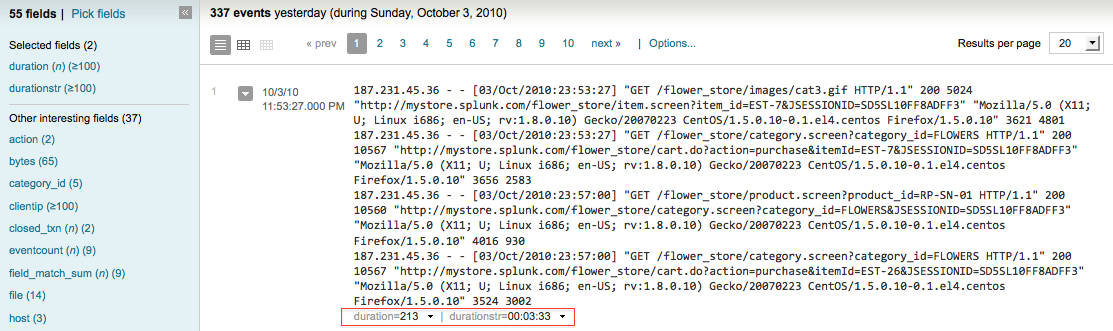
|  |
| --- |
| This example uses the sample dataset from [the Search Tutorial](http://docs.splunk.com/Documentation/Splunk/6.4.1/SearchTutorial/WelcometotheSearchTutorial) but should work with any format of Apache Web access log. Download the data set from [**this topic in the Search Tutorial**](http://docs.splunk.com/Documentation/Splunk/6.4.1/SearchTutorial/GetthetutorialdataintoSplunk) and follow the instructions to upload it to your Splunk deployment. 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:

[](http://docs.splunk.com/File:EvalEx4_eventsList.png)

**Answers**

Have questions? Visit [Splunk Answers](http://splunk-base.splunk.com/) and see what [questions and answers the Splunk community has using the eval command](http://splunk-base.splunk.com/tags/eval/).