splunk> Quick Reference Guide

CONCEPTS

Index-time and Search-time

During index-time processing, data is read from a source on a host and is classified into a source type. Timestamps are extracted, and the data is parsed into individual events. Line-breaking rules are applied to segment the events for display in search results. Each event is written to an index on disk, where it is later retrieved with a search request.

When a *search* starts, indexed events are retrieved from disk. *Fields* are extracted from the event's raw text. These events can then be transformed using the Splunk Enterprise search processing language to build *reports* and *visualizations* that can be added to dashboards.

Indexes

When data is added, Splunk Enterprise parses it into individual events, extracts the timestamp, applies line-breaking rules, and stores the events in an index. You can create new indexes for different inputs. By default, data is stored in the "main" index. Events are retrieved from one or more indexes during a search.

Events

An event is a set of values associated with a timestamp. It is a single entry of data and can have one or multiple lines. An event can be a text document, a configuration file, an entire stack trace, and so on. This is an example of an event in a web activity log:

```
173.26.34.223 - - [01/Jul/2009:12:05:27 -0700] "GET
/trade/app?action=logout HTTP/1.1" 200 2953
```

At search time, indexed events that match a specified search string can be categorized into event types. You can also define transactions to search for and group together events that are conceptually related but span a duration of time. Transactions can represent multistep business-related activity, such as all events related to a single customer session on a retail website.

Host

A *host* is the name of the physical or virtual device where an event originates. The host field provides an easy way to find all data originating from a specific device.

Source and Source Type

A *source* is the name of the file, directory, data stream, or other input from which a particular event originates. Sources are classified into *source types*, which can be either well known formats or defined by the user. Some familiar source types are HTTP web server logs and Windows event logs.

Events with the same source types can come from different sources. For example, events from the file `source=/var/log/messages` and from a syslog input port `source=UDP:514` often share the source type, `sourcetype=linux_syslog`.

Fields

Fields are searchable name and value pairings that distinguish one event from another because not all events have the same fields and field values. Using fields, you can write tailored searches to retrieve the specific events that you want and use the search commands. As Splunk Enterprise processes events at index-time and search-time, it extracts fields based on configuration file definitions and user-defined patterns.

Tags

Tags are aliases to particular field values. You can assign one or more tags to any field name/value combination, including event types, hosts, sources, and source types. Use tags to group related field values together or track abstract field values such as IP addresses or ID numbers by giving them more descriptive names.

SPLUNK ENTERPRISE FEATURES

Alerts

Alerts are triggered when conditions are met by search results for both historical and real-time searches. Alerts can be configured to trigger actions such as sending alert information to designated email addresses, post alert information to an RSS feed, and run a custom script, such as one that posts an "alert event" to syslog.

Data model

A data model is a hierarchically-structured search-time mapping of semantic knowledge about one or more datasets. It encodes the domain knowledge necessary to build a variety of specialized searches of those datasets. These specialized searches are in turn used by Splunk Enterprise to generate reports for Pivot users. Data model objects represent different datasets within the larger set of data indexed by Splunk Enterprise.

Pivot

Pivot refers to the table, chart, or data visualization you create using the Pivot Editor. The Pivot Editor enables users to map attributes defined by data model objects to a table or chart data visualization without having to write the searches to generate them. Pivots can be saved as reports and used to power dashboards.

Search

Search is the primary way users navigate data in Splunk Enterprise. You can write a search to retrieve events from an index, use statistical commands to calculate metrics and generate reports, search for specific conditions within a rolling time window, identify patterns in your data, predict future trends, and so on. Searches can be saved as reports and used to power dashboards.

Reports

Reports are saved searches and pivots. You can run reports on an adhoc basis, schedule them to run on a regular interval, set a scheduled report to generate alerts when the results of their runs meet particular conditions. Reports can be added to dashboards as dashboard panels.

Dashboards

Dashboards are made up of panels that contain modules such as search boxes, fields, charts, tables, forms, and so on. Dashboard panels are usually hooked up to saved searches or pivots. They can display the results of completed searches as well as data from backgrounded real-time searches.

SPLUNK ENTERPRISE COMPONENTS

Apps

Apps are a collection of configurations, knowledge objects, and customer designed views and dashboards that extend the Splunk Enterprise environment to fit the specific needs of organizational teams such as Unix or Windows system administrators, network security specialists, website managers, business analysts, and so on. A single Splunk Enterprise installation can run multiple apps simultaneously.

Forwarder and Receiver

A forwarder is a Splunk Enterprise instance that forwards data to another Splunk Enterprise instance (an indexer or another forwarder) or to a third party system. If the Splunk Enterprise instance (either an indexer or forwarder) is configured to receive data from a forwarder, it can also be called a receiver.

Indexer

An indexer is the Splunk Enterprise instance that indexes data. The indexer transforms the raw data into events and stores the events into an index. The indexer also searches the indexed data in response to search requests.

Search Head and Search Peer

In a distributed search environment, the search head is the Splunk Enterprise instance that directs search requests to a set of search peers and merges the results back to the user. The search peers are indexers that fulfill search requests from the search head. If the instance does only search and not indexing, it is usually referred to as a dedicated search head.

SEARCH PROCESSING LANGUAGE

A search is a series of commands and arguments. Commands are chained together with a pipe "|" character to indicate that the output of one command feeds into the next command on the right.

```
search | command arguments | command arguments | ...
```

At the start of the search pipeline, is an implied search command to retrieve events from the index. This search request can be written with keywords, quoted phrases, boolean expressions, wildcards, field name/value pairs, and comparison expressions.

See the following search example:

```
sourcetype=access_combined error | top 5 uri
```

This search retrieves indexed web activity events that contain the term "error" (ANDs are implied between search terms). For those events, it reports the top 5 most common URI values.

Searches commands are used to filter unwanted information, extract more information, calculate values, transform, and statistically analyze the indexed data. The search results retrieved from the index can be thought of as a dynamically created table. Each indexed event is a row and the field values are columns. Each search command redefines the shape of that table. For examples, search commands that filter events will remove rows, search commands that extract fields will add columns..

Subsearches

A subsearch is an argument to a command. A subsearch runs its own search and returns those results to the parent command as the argument value. A subsearch is contained in square brackets. For example, the following search uses a sub search to find all syslog events from the user that had the last login error:

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

Time Modifiers

Instead of using the custom time ranges in Splunk Web, you can specify a time range to retrieve events inline with your search by using the latest and earliest search modifiers. The relative times are specified with a string of characters that indicate the amount of time (integer and unit) and, optionally, a "snap to" time unit. The syntax for time modifiers is:

```
[+|-]<integer><unit>@<snap_time_unit>
```

The following search, "error earliest=-1d@d latest=-h@h" retrieves events containing "error" that occurred yesterday at midnight to the last hour, on the hour.

Time units are specified as seconds (s), minute (m), hour (h), day (d), week (w), month (mon), quarter (q), and year (y). The time integer defaults to 1. For example, "m" is the same as "1m".

Snapping rounds the time amount down to the latest time not after the specified time. For example, if it is 11:59:00 and you "snap to" hours (@h), the time will be 11:00:00 not 12:00:00. You can also "snap to" specific days of the week using @ w0 for Sunday, @w1 for Monday, and so on.

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

EVAL FUNCTIONS (continued)

FUNCTION	DESCRIPTION	EXAMPLES
searchmatch(X)	Returns true if the event matches the search string X.	`searchmatch("foo AND bar")`
split(X,"Y")	Returns X as a multi-valued field, split be delimiter Y.	`split(foo, ";")`
sqrt(X)	Returns the square root of X.	`sqrt(9)`
strftime(X,Y)	Returns epochtime value X rendered using the format specified by Y.	`strftime(_time, "%H:%M")`
strptime(X,Y)	Given a time represented by a string X, returns value parsed from format Y.	`strptime(timeStr, "%H:%M")`
substr(X,Y,Z)	Returns a substring field X from start position (1-based) Y for Z (optional) characters.	`substr("string", 1, 3)` `+substr("string", -3)`
time()	Returns the wall-clock time with microsecond resolution.	`time()`
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.	`tonumber("0A4",16)`

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:`
		`foo=615 and foo2=00:10:15:`
		`… | eval foo=615 | eval foo2 =`
		`tostring(foo, "duration")`

trim(X,Y)	Returns X with the characters in Y trimmed from both sides. If Y is not specified, spaces and tabs are trimmed.	`trim(" ZZZZabcZZ ", " Z")`

typeof(X)	Returns a string representation of its type.	`This example returns:`
		`"NumberStringBoolInvalid":`
		`typeof(12)+ typeof("string")+`
		`typeof(1==2)+ typeof(badfield)`

upper(X)	Returns the uppercase of X.	`upper(username)`
urldecode(X)	Returns the URL X decoded.	`urldecode("http%3A%2F%2Fwww.splunk.com%2Fdownload%3Fr%3Dheader")`
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.	`validate(isint(port), "ERROR: Port is not an integer", port >= 1 AND port <= 65535, "ERROR: Port is out of range")`

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 eval(field="value").
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, perc5(total) 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.

SEARCH EXAMPLES

Filter Results

Filter results to only include those with "fail" in their raw text and status=0.	`… | search fail status=0`
Remove duplicates of results with the same host value.	`… | dedup host`
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)"`

Group Results

Cluster results together, sort by their "cluster_count" values, and then return the 20 largest clusters (in data size).	`… | cluster t=0.9 showcount=true | sort limit=20 -cluster_count`
Group 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.	`… | transaction host cookie maxspan=30s maxpause=5s`
Group results with the same IP address (clientip) and where the first result contains "signon", and the last result contains "purchase".	`… | transaction clientip startswith="signon" endswith="purchase"`

Order Results

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

Reporting

Return events with uncommon values.	`… | anomalousvalue action=filter pthresh=0.02`
Return 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`
Return max(delay) for each value of foo split by the value of bar.	`… | chart max(delay) over foo by bar`
Return max(delay) for each value of foo.	`… | chart max(delay) over foo`
Remove all outlying numerical values.	`… | outlier`
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" (e.g., delay, xdelay, relay, etc).	`… | stats avg(*lay) by date_hour`
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`
Return the 20 most common values of the "url" field.	`… | top limit=20 url`
Return the least common values of the "url" field.	`… | rare url`

Add Fields

Set velocity to distance / time.	`… | eval velocity=distance/time`
Extract "from" and "to" fields using regular expressions. If a raw event contains "From: Susan To: David", then from=Susan and to=David.	`… | rex field=_raw "From: (?<from>.*) To: (?<to>.*)"`
Save the running total of "count" in a field called "total_count".	`… | accum count as total_count`
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`

Filter Fields

Keep the "host" and "ip" fields, and display them in the order: "host", "ip".	`… | fields + host, ip`
Remove the "host" and "ip" fields.	`… | fields - host, ip`

Modify Fields

Rename the "_ip" field as "IPAddress".	`… | rename _ip as IPAddress`
Change any host value that ends with "localhost" to "mylocalhost".	`… | replace *localhost with mylocalhost in host`

Multi-Valued Fields

Combine the multiple values of the recipients field into a single value	`… | nomv recipients`
Separate the values of the "recipients" field into multiple field values, displaying the top recipients	`… | makemv delim="," recipients | top recipients`
Create new results for each value of the multivalue field "recipients"	`… | mvexpand recipients`
For each result that is identical except for that RecordNumber, combine them, setting RecordNumber to be a multi-valued field with all the varying values.	`… | fields EventCode, Category, RecordNumber | mvcombine delim="," RecordNumber`
Find the number of recipient values	`… | eval to_count = mvcount(recipients)`
Find the first email address in the recipient field	`… | eval recipient_first = mvindex(recipient,0)`
Find all recipient values that end in .net or .org	`… | eval netorg_recipients = mvfilter(match(recipient, "\.net$") OR match(recipient, "\.org$"))`
Find the combination of the values of foo, "bar", and the values of baz	`… | eval newval = mvappend(foo, "bar", baz)`
Find the index of the first recipient value match "\.org$"	`… | eval orgindex = mvfind(recipient, "\.org$")`

Lookup Tables

Lookup the value of each event's 'user' field in the lookup table usertogroup, setting the event's 'group' field.	`… | lookup usertogroup user output group`
Write the search results to the lookup file "users.csv".	`… | outputlookup users.csv`
Read in the lookup file "users.csv" as search results.	`… | inputlookup users.csv`

REGULAR EXPRESSIONS (REGEXES)

Regular Expressions are useful in multiple areas: search commands regex and rex; eval functions match() and replace(); and in field extraction.

REGEX	NOTE	EXAMPLE	EXPLANATION
`\s`	white space	`\d\s\d`	digit space digit
`\S`	not white space	`\d\S\d`	digit non-whitespace digit
`\d`	digit	`\d\d\d-\d\d-\d\d\d\d`	SSN
`\D`	not digit	`\D\D\D`	three non-digits
`\w`	word character (letter, number, or _)	`\w\w\w`	three word chars
`\W`	not a word character	`\W\W\W`	three non-word chars
`[...]`	any included character	`[a-z0-9#]`	any char that is a thru z, 0 thru 9, or #
`[^...]`	no included character	`[^xyz]`	any char but x, y, or z
`*`	zero or more	`\w*`	zero or more words chars
`+`	one or more	`\d+`	integer
`?`	zero or one	`\d\d\d-?\d\d-?\d\d\d\d`	SSN with dashes being optional
`|`	or	`\w|\d`	word or digit character
`(?P<var> ...)`	named extraction	`(?P<ssn>\d\d\d-\d\d-\d\d\d\d)`	pull out a SSN and assign to 'ssn' field
`(?: ... )`	logical or atomic grouping	`(?:[a-zA-Z]|\d)`	alphabetic character OR a digit
`^`	start of line	`^\d+`	line begins with at least one digit
`$`	end of line	`\d+$`	line ends with at least one digit
`{...}`	number of repetitions	`\d{3,5}`	between 3-5 digits
`\`	escape	`\[`	escape the [char

COMMON SPLUNK STRPTIME FORMATS

strptime formats are useful for eval functions strftime() and strptime(), and for timestamping of event data.

Time	`%H`	24 hour (leading zeros) (00 to 23)
	`%I`	12 hour (leading zeros) (01 to 12)
	`%M`	Minute (00 to 59)
	`%S`	Second (00 to 61)
	`%N`	subseconds with width (%3N = millisecs, %6N = microsecs, %9N = nanosecs)
	`%p`	AM or PM
	`%Z`	Time zone (EST)
	`%z`	Time zone offset from UTC, in hour and minute: +hhmm or -hhmm. (-0500 for EST)
	`%s`	Seconds since 1/1/1970 (1308677092)
Days	`%d`	Day of month (leading zeros) (01 to 31)
	`%j`	Day of year (001 to 366)
	`%w`	Weekday (0 to 6)
	`%a`	Abbreviated weekday (Sun)
	`%A`	Weekday (Sunday)
Months	`%b`	Abbreviated month name (Jan)
	`%B`	Month name (January)
	`%m`	Month number (01 to 12)
Years	`%y`	Year without century (00 to 99)
	`%Y`	Year (2008)
Examples	`%Y-%m-%d`	1998-12-31
	`%y-%m-%d`	98-12-31
	`%b %d, %Y`	Jan 24, 2003
	`%B %d, %Y`	January 24, 2003
	`q|%d %b '%y = %Y-%m-%d|`	q|25 Feb '03 = 2003-02-25|

splunk>

Splunk Inc.
250 Brannan Street
San Francisco, CA 94107

www.splunk.com