

# Splunk Cheat Sheet

## SPLUNK CHEAT SHEET



## Brief Introduction of Splunk

The Internet of Things (IoT) and Internet of Bodies (IoB) generate much data, and searching for a needle of datum in such a haystack can be daunting.

Splunk is a Big Data mining tool. With Splunk, not only is it easier for users to excavate and analyze machine-generated data, but it also visualizes and creates reports on such data.

The screenshot shows the Splunk Enterprise search interface. The search bar contains the query: `source=tutorialdata.zip:* index=splunk_tutorial* | regex _raw!=""^d(1,3)\.d(1,3)\.d(1,3)\.d(1,3)* | search "code=A" | sort -_time`. The results show 1,908 events. The table below displays the first six events.

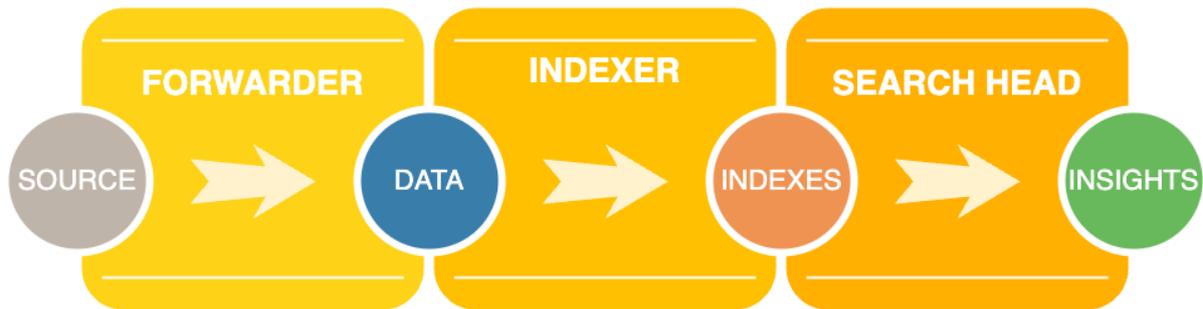
i	Time	Event
>	10/08/2022 18:00:00.000	[18/Aug/2022:18:00:00] VendorID=1006 Code=A AcctID=4507550694554937 host = vendor_sales source = tutorialdata.zip./vendor_sales/vendor_sales.log sourcetype = vendor_sales
>	10/08/2022 17:45:51.000	[18/Aug/2022:17:45:51] VendorID=7024 Code=A AcctID=4516434702861957 host = vendor_sales source = tutorialdata.zip./vendor_sales/vendor_sales.log sourcetype = vendor_sales
>	10/08/2022 17:38:28.000	[18/Aug/2022:17:38:28] VendorID=4115 Code=A AcctID=9954652649980119 host = vendor_sales source = tutorialdata.zip./vendor_sales/vendor_sales.log sourcetype = vendor_sales
>	10/08/2022 17:35:02.000	[18/Aug/2022:17:35:02] VendorID=4114 Code=A AcctID=6474571108617316 host = vendor_sales source = tutorialdata.zip./vendor_sales/vendor_sales.log sourcetype = vendor_sales
>	10/08/2022 17:34:44.000	[18/Aug/2022:17:34:44] VendorID=9112 Code=A AcctID=4968248489672714 host = vendor_sales source = tutorialdata.zip./vendor_sales/vendor_sales.log sourcetype = vendor_sales
>	10/08/2022 17:31:11.000	[18/Aug/2022:17:31:11] VendorID=7817 Code=A AcctID=5971388209753771

Splunk Enterprise search results on [sample data](#)

Splunk contains three processing components:

- The [Indexer](#) parses and indexes data [added](#) to Splunk.
- The [Forwarder](#) (optional) sends data from a source.

- The [Search Head](#) is for searching, analyzing, visualizing, and summarizing your data.



## Search Language in Splunk

Splunk uses what's called Search Processing Language (SPL), which consists of keywords, quoted phrases, Boolean expressions, wildcards (\*), parameter/value pairs, and comparison expressions. Unless you're joining two explicit Boolean expressions, omit the `AND` operator because Splunk assumes the space between any two search terms to be `AND`.

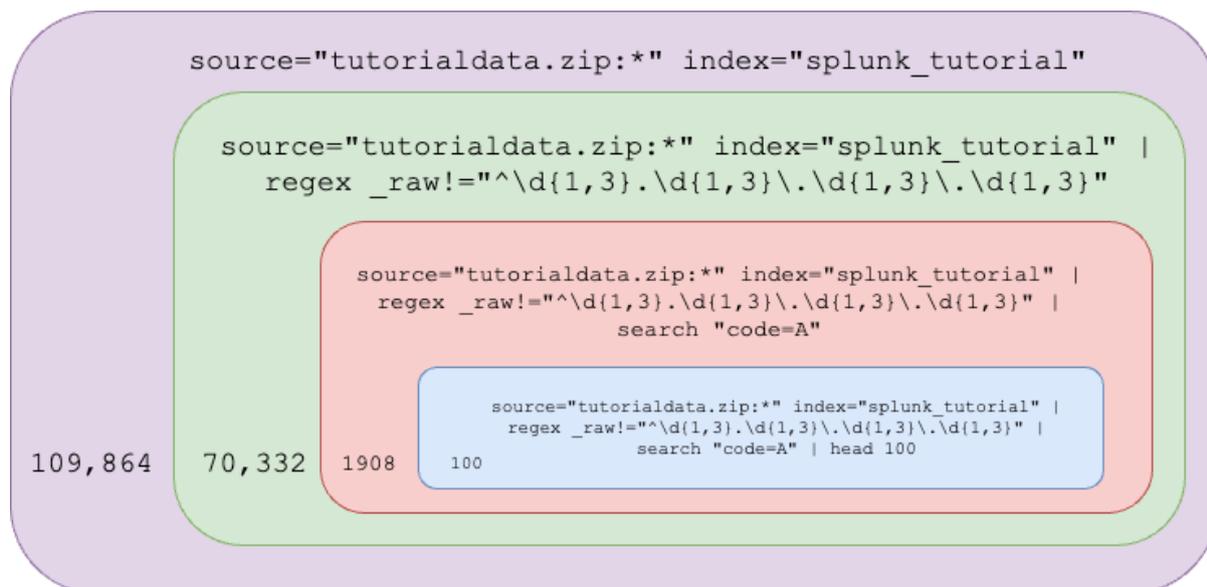
[Basic Search](#) offers a shorthand for simple keyword searches in a body of indexed data `myIndex` without further processing:

```
index=myIndex keyword
```

An event is an entry of data representing a set of values associated with a timestamp. It can be a text document, configuration file, or entire stack trace. Here is an example of an event in a web activity log:

```
[10/Aug/2022:18:23:46] userID=176 country=US paymentID=30495
```

Search commands help filter unwanted events, extract additional information, calculate values, transform data, and statistically analyze the indexed data. It is a process of narrowing the data down to your focus. Note the decreasing number of results below:



Finding entries without IPv4 address on [sample data](#)

## Common Search Commands

Command	Description
chart, timechart	Returns results in a tabular output for (time-series) charting
dedup X	Removes duplicate results on a field X
eval	Calculates an expression (see <a href="#">Calculations</a> )
fields	Removes fields from search results
head/tail N	Returns the first/last N results, where N is a positive integer
lookup	Adds field values from an external source
rename	Renames a field. Use wildcards (*) to specify multiple fields.
<a href="#">rex</a>	Extract fields according to specified regular expression(s)
search	Filters results to those that match the search expression
sort X	Sorts the search results by the specified fields X
stats	Provides statistics, grouped optionally by fields
mstats	Similar to stats but used on metrics instead of events
table	Displays data fields in table format.
top/rare	Displays the most/least common values of a field
transaction	Groups search results into <a href="#">transactions</a>
where	Filters search results using <a href="#">eval</a> expressions. For comparing two different fields.

## SPL Syntax

Begin by specifying the data using the parameter `index`, the equal sign `=`, and the data index of your choice: `index=index_of_choice`.

Complex queries involve the pipe character `|`, which feeds the output of the previous query into the next.

## Basic Search

This is the shorthand query to find the word `hacker` in an index called `cybersecurity`:  
`index=cybersecurity hacker`

SPL search terms	Description
<b>Full Text Search</b>	
<code>Cybersecurity</code>	Find the word "Cybersecurity" irrespective of capitalization
<code>White Black Hat</code>	Find those three words in any order irrespective of capitalization
<code>"White Black+Hat"</code>	Find the exact phrase with the given special characters, irrespective of capitalization
<b>Filter by fields</b>	
<code>source="/var/log/myapp/access.log" status=404</code>	All lines where the field <code>status</code> has value <code>404</code> from the file <code>/var/log/myapp/access.log</code>
<code>source="bigdata.rar:*" index="data_tutorial" Code=RED</code>	All entries where the field <code>Code</code> has value <code>RED</code> in the archive <code>bigdata.rar</code> indexed as <code>data_tutorial</code>
<code>index="customer_feedback" _raw="*excellent*"</code>	All entries whose text contains the keyword "excellent" in the indexed data set <code>customer_feedback</code>
<b>Filter by host</b>	
<code>host="myblog" source="/var/log/syslog" Fatal</code>	Show all <code>Fatal</code> entries from <code>/var/log/syslog</code> belonging to the blog host <code>myblog</code>
<b>Selecting an index</b>	
<code>index="myIndex" password</code>	Access the index called <code>myIndex</code> and text matching <code>password</code> .
<code>source="test_data.zip:*"</code>	Access the data archive called <code>test_data.zip</code> and parse all its entries (*).
<code>sourcetype="datasource01"</code>	(Optional) Search data sources whose type is <code>datasource01</code> .

This syntax also applies to the arguments following the `search` keyword. Here is an example of a longer SPL search string:

```
index=* OR index=_* sourcetype=generic_logs | search  
Cybersecurity | head 10000
```

In this example, `index=* OR index=_* sourcetype=generic_logs` is the data body on which Splunk performs `search Cybersecurity`, and then `head 10000` causes Splunk to show only the first (up to) 10,000 entries.

## Basic Filtering

You can filter your data using regular expressions and the Splunk keywords `rex` and `regex`. An example of finding deprecation warnings in the logs of an app would be:

```
index="app_logs" | regex error="Deprecation Warning"
```

SPL filters	Description	Examples
search	Find keywords and/or fields with given values	<ul style="list-style-type: none"> <li>index=names   search Chris</li> <li>index=emails   search emailAddr="*mysite.com"</li> </ul>
regex	Find expressions matching a given regular expression	Find logs not containing IPv4 addresses: index=syslogs   regex !="^\\d{1,3}.\\d{1,3}.\\d{1,3}.\\d{1,3}"
<a href="#">rex</a>	Extract fields according to specified regular expression(s) into a new field for further processing	Extract email addresses: source="email_dump.txt"   rex field=_raw "From: <(?(from>.*>)> To: <(?(to>.*>)>"

The biggest difference between `search` and `regex` is that you can only exclude query strings with `regex`. These two are equivalent:

- `source="access.log" Fatal`
- `source="access.log" | regex _raw=".*Fatal.*"`

But you can only use `regex` to find events that do not include your desired search term:

- `source="access.log" | regex _raw!=".*Fatal.*"`

The Splunk keyword `rex` helps determine the alphabetical codes involved in this dataset:

## New Search

```
source="tutorialdata.zip:*" index="splunk_tutorial" | rex field=_raw "Code=<{<Code>.*)" | dedup Code | table Code
```

✓ 14 events (before 11/08/2022 22:17:51.000) No Event Sampling ▾

Events Patterns **Statistics (14)** Visualization

50 Per Page ▾  Format Preview ▾

Code ▲

A

B

C

D

E

F

G

H

I

J

K

L

M

N

Alphabetical codes in [sample data](#)

## Calculations

Combine the following with `eval` to do computations on your data, such as finding the mean, longest and shortest comments in the following example:

```
index=comments | eval cmt_len=len(comment) | stats  
avg(cmt_len), max(cmt_len), min(cmt_len) by index
```

Function	Return value / Action	Usage: <code>eval foo=...</code>
<code>abs (X)</code>	absolute value of X	<code>abs (number)</code>
<code>case (X, "Y", ...)</code>	Takes pairs of arguments X and Y, where X arguments are Boolean expressions. When evaluated to TRUE, the arguments return the corresponding Y argument	<code>case(id == 0, "Amy", id == 1, "Brad", id == 2, "Chris")</code>
<code>ceil (X)</code>	Ceiling of a number X	<code>ceil(1.9)</code>
<code>cidrmatch ("X", Y)</code>	Identifies IP addresses that belong to a particular subnet	<code>cidrmatch("123.132.32.0/25", ip)</code>

<code>coalesce(X, ...)</code>	The first value that is not NULL	<code>coalesce(null(), "Returned val", null())</code> <code>n=cos(60) #1/2</code>
<code>cos(X)</code>	Cosine of X	
<code>exact(X)</code>	Evaluates an expression X using double precision floating point arithmetic	<code>exact(3.14*num)</code>
<code>exp(X)</code>	e (natural number) to the power X ( $e^X$ )	<code>exp(3)</code>
<code>if(X, Y, Z)</code>	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>
<code>in(field, value list)</code>	TRUE if a value in <code>valuelist</code> matches a value in <code>field</code> . You must use the <code>in()</code> function embedded inside the <code>if()</code> function	<code>if(in(status, "404", "500", "503"), "true", "false")</code>
<code>isbool(X)</code>	TRUE if X is Boolean	<code>isbool(field)</code>
<code>isint(X)</code>	TRUE if X is an integer	<code>isint(field)</code>
<code>isnull(X)</code>	TRUE if X is NULL	<code>isnull(field)</code>
<code>isstr(X)</code>	TRUE if X is a string	<code>isstr(field)</code>
<code>len(X)</code>	Character length of string X	<code>len(field)</code>
<code>like(X, "Y")</code>	TRUE if and only if X is like the SQLite pattern in Y	<code>like(field, "addr%")</code>
<code>log(X, Y)</code>	Logarithm of the first argument X where the second argument Y is the base. Y defaults to 10 (base-10 logarithm)	<code>log(number, 2)</code>
<code>lower(X)</code>	Lowercase of string X	<code>lower(username)</code>
<code>ltrim(X, Y)</code>	X with the characters in Y trimmed from the left side. Y defaults to spaces and tabs	<code>ltrim(" ZZZabcZZ ", "Z")</code>
<code>match(X, Y)</code>	TRUE if X matches the regular expression pattern Y	<code>match(field, "^\d{1,3}\.\d\$")</code>
<code>max(X, ...)</code>	The maximum value in a series of data X, ...	<code>max(delay, mydelay)</code>
<code>md5(X)</code>	MD5 hash of a string value X	<code>md5(field)</code>
<code>min(X, ...)</code>	The minimum value in a series of data X, ...	<code>min(delay, mydelay)</code>
<code>mvcount(X)</code>	Number of values of X	<code>mvcount(multifield)</code>
<code>mvfilter(X)</code>	Filters a multi-valued field based on the Boolean expression X	<code>mvfilter(match(email, "net\$"))</code>
<code>mvindex(X, Y, Z)</code>	Returns a subset of the multi-valued field X from start position (zero-based) Y to Z (optional)	<code>mvindex(multifield, 2)</code>
<code>mvjoin(X, Y)</code>	Joins the individual values of a multi-valued field X using string delimiter Y	<code>mvjoin(address, ";")</code>
<code>now()</code>	Current time as Unix timestamp	<code>now()</code>

<code>null()</code>	NULL value. This function takes no arguments.	<code>null()</code>
<code>nullif(X,Y)</code>	X if the two arguments, fields X and Y, are different. Otherwise returns NULL.	<code>nullif(fieldX, fieldY)</code>
<code>random()</code>	Pseudo-random number ranging from 0 to 2147483647	<code>random()</code>
<code>relative_time(X,Y)</code>	Unix timestamp value of relative time specifier Y applied to Unix timestamp X	<code>relative_time(now(), "-1d@d")</code>
<code>replace(X,Y,Z)</code>	A string formed by substituting string Z for every occurrence of regex string Y in string X  The example swaps the month and day numbers of a date.	<code>replace(date, "^(\d{1,2})/(\d{1,2})/", "\2/\1/")</code>
<code>round(X,Y)</code>	X rounded to the number of decimal places specified by Y, or to an integer for omitted Y	<code>round(3.5)</code>
<code>rtrim(X,Y)</code>	X with the characters in (optional) Y trimmed from the right side. Trim spaces and tabs for unspecified Y	<code>rtrim(" ZZZZabcZ", "Z")</code>
<code>split(X,"Y")</code>	X as a multi-valued field, split by delimiter Y	<code>split(address, ";")</code>
<code>sqrt(X)</code>	Square root of X	<code>sqrt(9) # 3</code>
<code>strftime(X,Y)</code>	Unix timestamp value X rendered using the format specified by Y	<code>strftime(time, "%H:%M")</code>
<code>strptime(X,Y)</code>	Value of Unix timestamp X as a string parsed from format Y	<code>strptime(timeStr, "%H:%M")</code>
<code>substr(X,Y,Z)</code>	Substring of X from start position (1-based) Y for (optional) Z characters	<code>substr("string", 1, 3) #str</code>
<code>time()</code>	Current time to the microsecond.	<code>time()</code>
<code>tonumber(X,Y)</code>	Converts input string X to a number of numerical base Y (optional, defaults to 10)	<code>tonumber("FF", 16)</code>
<code>tostring(X,Y)</code>	Field value of X as a string.  If X is a number, it reformats it as a string. If X is a Boolean value, it reformats to "True" or "False" strings.  If X is a number, the optional second argument Y is one of: <ul style="list-style-type: none"> <li>- "hex": convert X to hexadecimal,</li> <li>- "commas": formats X with commas and two decimal places, or</li> <li>- "duration": converts seconds X to readable time format HH:MM:SS.</li> </ul>	This example returns <code>bar=00:08:20:</code>  <code>  makeresults   eval bar = tostring(500, "duration")</code>
<code>typeof(X)</code>	String representation of the field type	This example returns <code>"NumberBool":</code>  <code>  makeresults   eval n=typeof(12) + typeof(1==2)</code>

<code>urldecode(X)</code>	URL X, decoded.	<code>urldecode("http%3A%2F%2Fwww.site.com%2Fview%3Fr%3Dabout")</code>
<code>validate(X, Y, ...)</code>	For pairs of Boolean expressions X and strings Y, returns the string Y corresponding to the first expression X which evaluates to False, and defaults to NULL if all X are True.	<code>validate(isint(N), "Not an integer", N&gt;0, "Not positive")</code>

## Statistical and Graphing Functions

Common statistical functions used with the `chart`, `stats`, and `timechart` commands.

Field names can contain wildcards (\*), so `avg(*delay)` might calculate the average of the `delay` and `*delay` fields.

Function	Return value <b>Usage:</b> <code>stats foo=...</code> / <code>chart bar=...</code> / <code>timechart t=...</code>
<code>avg(X)</code>	average of the values of field X
<code>count(X)</code>	number of occurrences of the field X. To indicate a specific field value to match, format X as <code>eval(field="desired_value")</code> .
<code>dc(X)</code>	count of distinct values of the field X
<code>earliest(X)</code> <code>latest(X)</code>	chronologically earliest/latest seen value of X
<code>max(X)</code>	maximum value of the field X. For non-numeric values of X, compute the max using alphabetical ordering.
<code>median(X)</code>	middle-most value of the field X
<code>min(X)</code>	minimum value of the field X. For non-numeric values of X, compute the min using alphabetical ordering.
<code>mode(X)</code>	most frequent value of the field X
<code>percN(Y)</code>	N-th percentile value of the field Y. N is a non-negative integer < 100. Example: <code>perc50(total)</code> = 50th percentile value of the field <code>total</code> .
<code>range(X)</code>	difference between the max and min values of the field X
<code>stdev(X)</code>	sample standard deviation of the field X
<code>stdevp(X)</code>	population standard deviation of the field X
<code>sum(X)</code>	sum of the values of the field X
<code>sumsq(X)</code>	sum of the squares of the values of the field X
<code>values(X)</code>	list of all distinct values of the field X as a multi-value entry. The order of the values is alphabetical
<code>var(X)</code>	sample variance of the field X

## Index Statistics

Compute index-related statistics.

From this point onward, `splunk` refers to the partial or full path of the Splunk app on your device `$SPLUNK_HOME/bin/splunk`, such as `/Applications/Splunk/bin/splunk` on macOS, or, if you have performed `cd` and entered `/Applications/Splunk/bin/`, simply `./splunk`.

Function	Description
<code>  eventcount summarize=false index=*   dedup index   fields index</code>	List all indexes on your Splunk instance. On the command line, use this instead: <code>splunk list index</code>
<code>  eventcount summarize=false report_size=true index=*   eval size_MB = round(size_bytes/1024/1024,2)</code>	Show the number of events in your indexes and their sizes in MB and bytes
<code>  REST /services/data/indexes   table title currentDBSizeMB</code>	List the titles and current database sizes in MB of the indexes on your Indexers
<code>index=_internal source=*metrics.log group=per_index_thruput series=*   eval MB = round(kb/1024,2)   timechart sum(MB) as MB by series</code>	Query write amount in MB per index from <code>metrics.log</code>
<code>index=_internal metrics kb series!=_* "group=per_host_thruput"   timechart fixedrange=t span=1d sum(kb) by series</code>	Query write amount in KB per day per Indexer by each host
<code>index=_internal metrics kb series!=_* "group=per_index_thruput"   timechart fixedrange=t span=1d sum(kb) by series</code>	Query write amount in KB per day per Indexer by each index

## Reload apps

To reload Splunk, enter the following in the address bar or command line interface.

Address bar	Description
<code>http://localhost:8000/debug/refresh</code>	Reload Splunk. Replace <code>localhost:8000</code> with the base URL of your Splunk Web server if you're not running it on your local machine.
Command line	Description
<code>splunk _internal call /data/inputs/monitor/_reload</code>	Reload Splunk file input configuration
<code>splunk stop splunk enable webserver splunk start</code>	These three lines in succession restart Splunk.

## Debug Traces

You can enable traces listed in `$SPLUNK_HOME/var/log/splunk/splunkd.log`.

To change trace topics permanently, go to `$$SPLUNK_HOME/bin/splunk/etc/log.cfg` and change the trace level, for example, from INFO to DEBUG:  
`category.TcpInputProc=DEBUG`

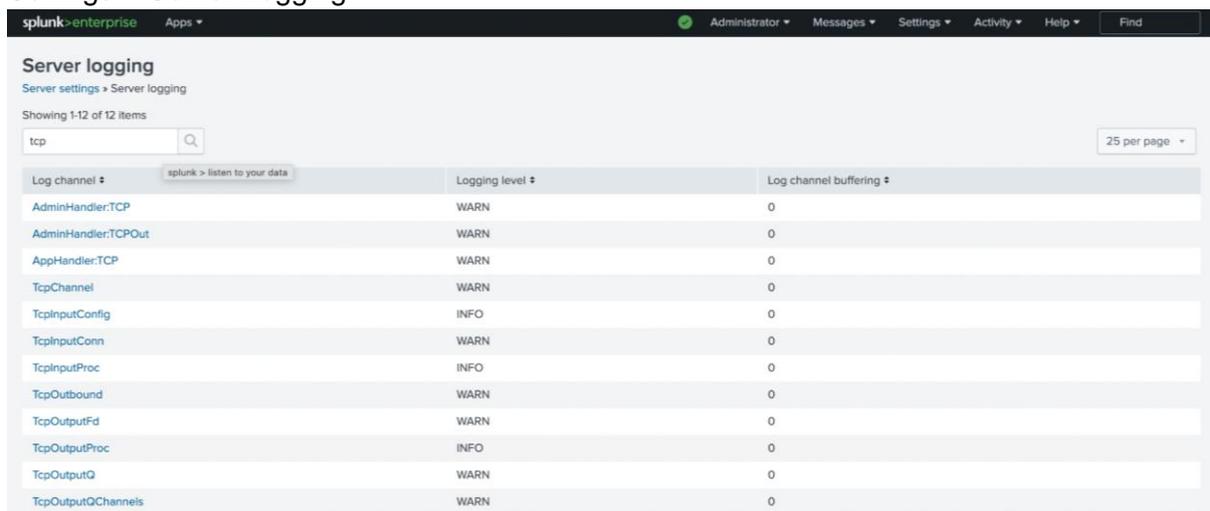
Then

```
08-10-2022 05:20:18.653 -0400 INFO ServerConfig [0  
MainThread] - Will generate GUID, as none found on this  
server.
```

becomes

```
08-10-2022 05:20:18.653 -0400 DEBUG ServerConfig [0  
MainThread] - Will generate GUID, as none found on this  
server.
```

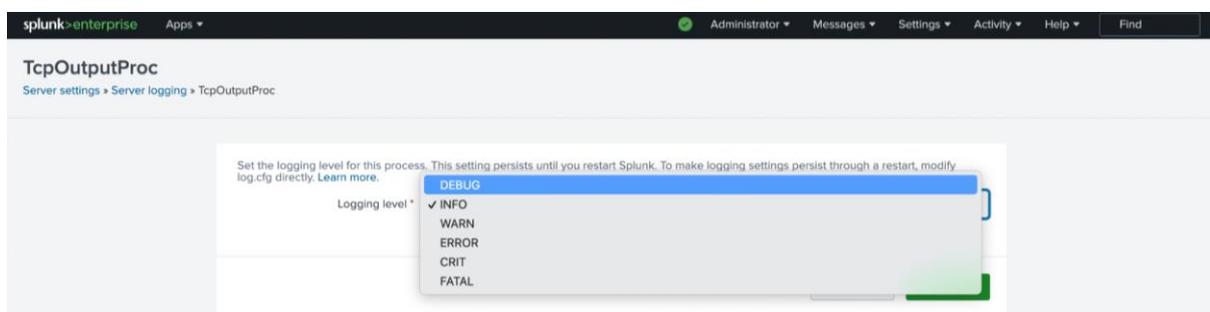
To change the trace settings only for the current instance of Splunk, go to Settings > Server Settings > Server Logging:



The screenshot shows the 'Server logging' configuration page in Splunk. A search filter 'tcp' is applied to the table. The table lists various log channels with their current logging levels and buffering settings.

Log channel	Logging level	Log channel buffering
AdminHandler:TCP	WARN	0
AdminHandler:TCPOut	WARN	0
AppHandler:TCP	WARN	0
TcpChannel	WARN	0
TcpInputConfig	INFO	0
TcpInputConn	WARN	0
TcpInputProc	INFO	0
TcpOutbound	WARN	0
TcpOutputFd	WARN	0
TcpOutputProc	INFO	0
TcpOutputQ	WARN	0
TcpOutputQChannels	WARN	0

Filter the log channels as above.



Select your new log trace topic and click Save. This persists until you stop the server.

## Configuration

The following changes Splunk settings. Where necessary, append `-auth user:pass` to the end of your command to authenticate with your Splunk web server credentials.

Command line	Description
<a href="#">Troubleshooting</a>	
<code>splunk btool inputs list</code>	List Splunk configurations
<code>splunk btool check</code>	Check Splunk configuration syntax
<a href="#">Input management</a>	
<code>splunk _internal call /data/inputs/tcp/raw</code>	List TCP inputs
<code>splunk _internal call /data/inputs/tcp/raw -get:search sourcetype=foo</code>	Restrict listing of TCP inputs to only those with a source type of <code>foo</code>
License details of your current Splunk instance	
<code>splunk list licenses</code>	Show your current license
User management	
<code>splunk _internal call /authentication/providers/services/_reload</code>	Reload authentication configurations for Splunk 6.x
<code>splunk _internal call /services/authentication/users -get:search admin</code>	Search for all users who are admins
<code>splunk _internal call /services/authentication/users -get:search indexes_edit</code>	See which users could edit indexes
<code>splunk _internal call /services/authentication/users /helpdesk -method DELETE</code>	Use the remove link in the returned XML output to delete the user <code>helpdesk</code>

## Capacity Planning

Importing large volumes of data takes much time. If you're using Splunk in-house, the software installation of Splunk Enterprise alone requires ~2GB of disk space. You can find an excellent online calculator at [splunk-sizing.appspot.com](http://splunk-sizing.appspot.com).

The essential factors to consider are:

- Input data
  - Specify the amount of data concerned. The more data you send to Splunk Enterprise, the more time Splunk needs to index it into results that you can search, report and generate alerts on.
- Data Retention
  - Specify how long you want to keep the data. You can only keep your imported data for a maximum length of 90 days or approximately three months.
  - Hot/Warm: short-term, in days.
  - Cold: mid-term, in weeks.
  - Archived (Frozen): long-term, in months.
- Architecture
  - Specify the number of nodes required. The more data to ingest, the greater the number of nodes required. Adding more nodes will improve indexing throughput and search performance.

- Storage Required
  - Specify how much space you need for hot/warm, cold and archived data storage.
- Storage Configuration
  - Specify the location of the storage configuration. If possible, spread each type of data across separate volumes to improve performance: hot/warm data on the fastest disk, cold data on a slower disk, and archived data on the slowest.

We hope this Splunk cheat sheet makes Splunk a more enjoyable experience for you. To download a PDF version of this Splunk cheat sheet, click [here](#).