search

Description

Use the `search` command to retrieve events from indexes or filter the results of a previous search command in the pipeline. You can retrieve events from your indexes, using keywords, quoted phrases, wildcards, and field-value expressions. The `search` command is implied at the beginning of any search. You do not need to specify the `search` command at the beginning of your search criteria.

When the `search` command is not the first command in the pipeline, the `search` command is used to filter the results of the previous command.

The `search` command can also be used in a subsearch. See about subsearches in the `Search Manual`.

After you retrieve events, you can apply commands to transform, filter, and report on the events. Use the vertical bar ( | ) , or pipe character, to apply a command to the retrieved events.

Syntax

`search <logical-expression>`

**Required arguments**

`<logical-expression>`

**Syntax:** `<logical-expression> | <time-opts> | <search-modifier> | NOT <logical-expression> | <index-expression> | <comparison-expression> | <logical-expression> [OR] <logical-expression>`

**Description:** Includes all keywords or field-value pairs used to describe the events to retrieve from the index. Include parenthesis as necessary. Use Boolean expressions, comparison operators, time modifiers, search modifiers, or combinations of expressions for this argument.

**Logical expression options**

`<comparison-expression>`

**Syntax:** `<field><comparison-operator><value> | <field> IN (<value-list>)`

**Description:** Compare a field to a literal value or provide a list of values that can appear in the 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. See Time options.

Comparison expression options

<comparison-operator>
Syntax: = | != | < | <= | > | >=
Description: You can use comparison operators when searching field/value pairs. Comparison expressions with the equal ( = ) or not equal ( != ) operator compare string values. For example, "1" does not match "1.0". Comparison expressions with greater than or less than operators < > <= >= numerically compare two numbers and lexicographically compare other values. See Usage.

<field>
Syntax: <string>
Description: The name of a field.

<value>
Syntax: <literal-value>
Description: In comparison-expressions, the literal number or string value of a field.

<value-list>
Syntax: (<literal-value>, <literal-value>, ...)
Description: Used with the IN operator to specify two or more values. For example use error IN (400, 402, 404, 406) instead of error=400 OR error=402 OR error=404 OR error=406

Index expression options

<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 software searches the _raw field for the matching events
or results.

<search-modifier>
  Syntax: <sourcetype-specifier> | <host-specifier> | <hosttag-specifier> | <source-specifier> | <savedsplunk-specifier> | <eventtype-specifier> | <eventtypetag-specifier> | <splunk_server-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>.
    ◊ Read more about searching with default fields in the Knowledge Manager manual.
    ◊ Read more about using tags and field aliases in the Knowledge Manager manual.

<sourcetype-specifier>
  Syntax: sourcetype=<string>
  Description: Search for events from the specified sourcetype field.

<host-specifier>
  Syntax: host=<string>
  Description: Search for events from the specified host field.

<hosttag-specifier>
  Syntax: hosttag=<string>
  Description: Search for events that have hosts that are tagged by the string.

<eventtype-specifier>
  Syntax: eventtype=<string>
  Description: Search for events that match the specified event type.

<eventtypetag-specifier>
  Syntax: eventtypetag=<string>
  Description: Search for events that would match all eventtypes tagged by the string.

<savedsplunk-specifier>
  Syntax: savedsearch=<string> | savedsplunk=<string>
  Description: Search for events that would be found by the specified saved search.

<source-specifier>
**Syntax:** source=<string>
**Description:** Search for events from the specified source field.

<splunk_server-specifier>
**Syntax:** splunk_server=<string>
**Description:** Search for events from a specific server. Use "local" to refer to the search head.

**Time options**

For a list of time modifiers, see Time modifiers for search.

<timeformat>
**Syntax:** timeformat=<string>
**Description:** Set the time format for starttime and endtime terms.
**Default:** 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. For more about this time modifier syntax, see Specify time modifiers in your search 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.

**Usage**

The `search` command is an event-generating command when it is the first command in the search, before the first pipe. When the `search` command is used further down the pipeline, it is a distributable streaming command. See Command types.
When `search` is the first command in the search, you can use terms such as keywords, phrases, fields, boolean expressions, and comparison expressions to specify exactly which events you want to retrieve from Splunk indexes.

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

To search field values that are SPL operators or keywords, such as `country=IN`, `country=AS`, `iso=AND`, or `state=OR`, you must enclose the operator or keyword in quotation marks. For example: `country="IN"`.

See Use the search command in the *Search Manual*.

**Boolean expressions**

The order in which Boolean expressions are evaluated with the `search` is:

1. Expressions within parentheses
2. NOT clauses
3. OR clauses
4. AND clauses

This evaluation order is different than the order used with the `where` command. The `where` command evaluates AND clauses before OR clauses.

**Comparing two fields**

To compare two fields, **do not specify** `index=myindex fieldA=fieldB` or `index=myindex fieldA!=fieldB` with the `search` command. When specifying a comparison expression, the `search` command expects a `<field>` compared with a `<value>`. The `search` command interprets `fieldB` as the value, and not as the name of a field.

Use the `where` command to compare two fields.

```
index=myindex | where fieldA=fieldB
```
For not equal comparisons, you can specify the criteria in several ways.

\[
\text{index=myindex} \mid \text{where fieldA}\neq\text{fieldB}
\]

or

\[
\text{index=myindex} \mid \text{where NOT fieldA=fieldB}
\]

See Difference between NOT and \(!=\) in the Search Manual.

**Multiple field-value comparisons with the IN operator**

Use the IN operator when you want to determine if a field contains one of several values.

For example, use this syntax:

\[
\ldots \text{error_code IN (400, 402, 404, 406)} \mid \ldots
\]

Instead of this syntax:

\[
\ldots \text{error_code}=400 \text{ OR error_code}=402 \text{ OR error_code}=404 \text{ OR error_code}=406 \mid \ldots
\]

When used with the search command, you can use a wildcard character in the list of values for the IN operator. For example:

\[
\ldots \text{error_code IN (40*)} \mid \ldots
\]

You can use the NOT operator with the IN operator. For example:

\[
\ldots \text{NOT clientip IN (211.166.11.101, 182.236.164.11, 128.241.220.82)} \mid \ldots
\]

There is also an IN function that you can use with the eval and where commands. Wild card characters are not allowed in the values list when the IN function is used with the eval and where commands. See Comparison and Conditional functions.

**Lexicographical order**

Lexicographical order sorts items based on the values used to encode the items in computer memory. In Splunk software, this is almost always UTF-8 encoding, which is a superset of ASCII.
• Numbers are sorted before letters. Numbers are sorted based on the first digit. For example, the numbers 10, 9, 70, 100 are sorted lexicographically as 10, 100, 70, 9.
• Uppercase letters are sorted before lowercase letters.
• Symbols are not standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

You can specify a custom sort order that overrides the lexicographical order. See the blog Order Up! Custom Sort Orders.

**Quotes and escaping characters**

In general, you need quotation marks around phrases and field values that include white spaces, commas, pipes, quotations, and brackets. Quotation marks must be balanced. An opening quotation must be followed by an unescaped closing quotation. 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 quotation marks around keywords and phrases if you do not 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 quotation marks. 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 quotation mark 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 to force Splunk software to match whatever is inside the parentheses as a single term in the index. `TERM` is more useful when the term contains minor segmenters, such as periods, and is bounded by major segmenters, such as spaces or commas. In fact, `TERM` does not work for terms that are not bounded by major breakers.

See Use CASE and TERM to match phrases in the Search Manual.

**Search with `CASE()`**

You can use the `CASE()` directive to search for terms and field values that are case-sensitive.

See Use CASE and TERM to match phrases in the Search Manual.

**Examples**

These examples demonstrate how to use the `search` command. You can find more examples in the Start Searching topic of the Search Tutorial.

1. **Field-value pair matching**

   This example demonstrates field-value pair matching for specific values of source IP (src) and destination IP (dst).

   \[ src="10.9.165.*" \text{ OR } dst="10.9.165.8" \]

2. **Using boolean and comparison operators**

   This example demonstrates field-value pair matching with boolean and comparison operators. Search for events with code values of either 10 or 29, and any host that isn't "localhost", and an \texttt{xqp} value that is greater than 5.
(code=10 OR code=29) host!="localhost" xqp>5

In this example you could also use the IN operator since you are specifying two field-value pairs on the same field. The revised search is:

code IN(10, 29) host!="localhost" xqp>5

3. Using wildcards

This example demonstrates field-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*)

In this example you could also use the IN operator since you are specifying two field-value pairs on the same field. The revised search is:

host=webserver* status IN(4*, 5*)

4. Using the IN operator

This example shows how to use the IN operator to specify a list of field-value pair matchings. In the events from an access.log file, search the action field for the values addtocart or purchase.

sourcetype=access_combined_wcookie action IN (addtocart, purchase)

5. Specifying a secondary search

This example uses the search command twice. The search command is implied at the beginning of every search with the criteria eventtype=web-traffic. The search command is used again later in the search pipeline to filter out the 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 | transaction clientip startswith="login" endswith="logout" | search eventcount>3

6. Using the NOT or != comparisons

Searching with the boolean "NOT" comparison operator is not the same as using the "!=" comparison.
The following search returns everything except fieldA="value2", including all other fields.

NOT fieldA="value2"

The following search returns events where fieldA exists and does not have the value "value2".

fieldA!="value2"

If you use a wildcard for the value, NOT fieldA=* returns events where fieldA is null or undefined, and fieldA!=* never returns any events.

See Difference between NOT and != in the Search Manual.

Answers

Have questions? Visit Splunk Answers and see what questions and answers the Splunk community has using the search command.