**dedup**

**Description**

Removes the events that contain an identical combination of values for the fields that you specify.

With the `dedup` command, you can specify the number of duplicate events to keep for each value of a single field, or for each combination of values among several fields. Events returned by `dedup` are based on search order. For historical searches, the most recent events are searched first. For real-time searches, the first events that are received are searched, which are not necessarily the most recent events.

You can specify the number of events with duplicate values, or value combinations, to keep. You can sort the fields, which determines which event is retained. Other options enable you to retain events with the duplicate fields removed, or to keep events where the fields specified do not exist in the events.

**Syntax**

`dedup [<int>] <field-list> [keepevents=<bool>] [keepempty=<bool>] [consecutive=<bool>] [sortby <sort-by-clause>]`

*Required arguments*

- `<field-list>`
  - **Syntax:** `<string> <string> ...`
  - **Description:** A list of field names to remove duplicate values from.

*Optional arguments*

- `consecutive`
  - **Syntax:** `consecutive=<bool>`
  - **Description:** If true, only remove events with duplicate combinations of values that are consecutive.
  - **Default:** false

- `keepempty`
  - **Syntax:** `keepempty=<bool>`
  - **Description:** If set to true, keeps every event where one or more of the specified fields is not present (null).
Default: false. All events where any of the selected fields are null are dropped.

The keepempty=true argument keeps every event that does not have one or more of the fields in the field list. To keep N representative events for combinations of field values including null values, use the fillnull command to provide a non-null value for these fields. For example:

```plaintext
...| fillnull value="MISSING" field1 field2 | dedup field1 field2
```

keepevents

**Syntax:** keepevents=<bool>

**Description:** If true, keep all events, but will remove the selected fields from events after the first event containing a particular combination of values.

**Default:** false. Events are dropped after the first event of each particular combination.

<N>

**Syntax:** <int>

**Description:** The dedup command retains multiple events for each combination when you specify N. The number for N must be greater than 0. If you do not specify a number, only the first occurring event is kept. All other duplicates are removed from the results.

<sort-by-clause>

**Syntax:** sortby ( - | + ) <sort-field> [(- | +) <sort_field> ...]

**Description:** List of the fields to sort by and the sort order. Use the dash symbol ( - ) for descending order and the plus symbol ( + ) for ascending order. You must specify the sort order for each field specified in the <sort-by-clause>. The <sort-by-clause> determines which of the duplicate events to keep. When the list of events is sorted, the top-most event in the sorted list is retained.

**Sort field options**

**Syntax:** <field> | auto(<field>) | str(<field>) | ip(<field>) | num(<field>)

**Description:** The options that you can specify to sort the events.

**Syntax:** <string>
**Description:** The name of the field to sort.

```
auto
```

**Syntax:** `auto(<field>)`

**Description:** Determine automatically how to sort the field values.

```
ip
```

**Syntax:** `ip(<field>)`

**Description:** Interpret the field values as IP addresses.

```
num
```

**Syntax:** `num(<field>)`

**Description:** Interpret the field values as numbers.

```
str
```

**Syntax:** `str(<field>)`

**Description:** Order the field values by using the lexicographic order.

---

**Usage**

The `dedup` command is a **streaming command** or a **dataset processing command**, depending on which arguments are specified with the command. See Command types.

Avoid using the `dedup` command on the `_raw` field if you are searching over a large volume of data. If you search the `_raw` field, the text of every event in memory is retained which impacts your search performance. This is expected behavior. This behavior applies to any field with high cardinality and large size.

**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.
Examples

Example 1:
Remove duplicates of results with the same 'host' value.

... | dedup host

Example 2:
Remove duplicates of results with the same 'source' value and sort the events by
the '_time' field in ascending order.

... | dedup source sortby +_time

Example 3:
Remove duplicates of results with the same 'source' value and sort the events by
the '_size' field in descending order.

... | dedup source sortby --size

Example 4:
For events that have the same 'source' value, keep the first 3 that occur and
remove all subsequent events.

... | dedup 3 source

Example 5:
For events that have the same 'source' AND 'host' values, keep the first 3 that
occur and remove all subsequent events.

... | dedup 3 source host

See also

uniq
Answers

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