

Lab Report — Search Commands Deep Dive: Part 2

Name: Dhruvish

Date: October 25, 2025

Platform: Splunk Cloud Trial

Objective

This lab focused on expanding your Splunk Search Processing Language (SPL) skills using **eval-based functions** and **string manipulation helpers**.

You practiced using if(), case(), coalesce(), lower()/upper(), and len() to:

- Label and classify events
- Normalize inconsistent field values
- Create cleaner, more informative summaries

These functions are essential for refining data, preparing reports, and ensuring consistent field logic in dashboards and alerts.

Tools Used

- **Splunk Cloud Trial** (or Splunk Free)
 - **Web Browser**
-

Procedure and Observations

Step 1: Dataset & Time Setup

- Opened **Apps ▶ Search & Reporting**.
- Set time range to **Last 24 hours**.
- Used the built-in dataset:
`index=_internal sourcetype=splunkd_ui_access`
- Confirmed the presence of key fields: status, user, method, uri_path.

Step 2: Label Events with if()

Used conditional logic to label each event as a “Success” or “Non-200” based on HTTP status.

```
index=_internal sourcetype=splunkd_ui_access
```

```
| eval outcome=if(status=200,"Success","Non-200")
```

```
| stats count by outcome
```

Then filtered only failed outcomes:

```
... | where outcome="Non-200"
```

Step 3: Multi-Branch Logic with case()

Created HTTP class categories using case() for multi-condition evaluation:

```
index=_internal sourcetype=splunkd_ui_access
```

```
| eval http_class=case(
```

```
status<200,"Informational",
```

```
status>=200 AND status<300,"Success",
```

```
status=404,"Not Found",
```

```

status>=400 AND status<500,"Client Error",
status>=500,"Server Error",
true(),"Other")
| stats count by http_class | sort - count

```

This allowed classification of status codes into meaningful HTTP response classes.

http_class	count
Success	225
Other	6
Client_Error	4
Not_Found	3

Step 4: Normalize Identities with coalesce()

Unified potentially inconsistent user fields using coalesce() to select the first available non-null value.

index=_internal sourcetype=splunkd_ui_access

```

| eval user_norm=coalesce(user, user_name, "-unknown-")
| stats count by user_norm | sort - count

```

Then joined with class labeling:

```

... | eval http_class=case(status>=200 AND status<300,"Success",
status>=500,"Server Error",
true(),"Other")

```

| stats count by user_norm http_class | sort - count

user_norm	http_class	count
-	Other	1
admin	Other	1

Step 5: Clean Keys with lower()/upper()

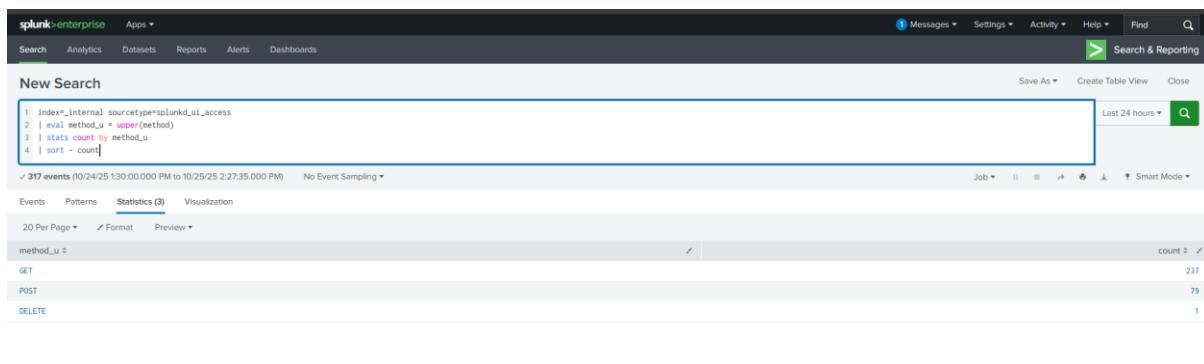
Standardized usernames and methods to consistent casing for aggregation.

- **Lowercasing usernames:**

```
index=_internal sourcetype=splunkd_ui_access  
| eval user_norm=lower(coalesce(user,user_name))  
| stats dc(uri_path) as unique_paths by user_norm | sort - unique_paths
```

- **Uppercasing HTTP methods:**

```
... | eval method_u=upper(method)  
| stats count by method_u
```



Step 6: Spot Long URLs with len()

Used len() to measure the length of URI paths and identify outliers.

```
index=_internal sourcetype=splunkd_ui_access
```

```
| eval url_len=len(uri_path)  
| stats max(url_len) as max_len, avg(url_len) as avg_len by user  
| sort - max_len
```

Filtered for URLs longer than 60 characters:

```
index=_internal sourcetype=splunkd_ui_access
```

```
| eval url_len=len(uri_path)  
| where url_len > 60  
| table _time user method uri_path url_len status
```

[Insert SS #6: Table showing long URL results (where url_len > 60)]

_time	user	method	uri_path	url_len	status
2025-10-25 14:23:14.463	admin	GET	/en-US/splunkd/_raw/servicesNs/nobody/search/search/v2/jobs/rt_md_1761382392.309	81	200
2025-10-25 14:23:12.651	admin	GET	/en-US/splunkd/_raw/servicesNs/nobody/splunk_instrumentation/instrumentation_controller/instrumentation_eligibility	116	200
2025-10-25 14:23:13.888	admin	GET	/en-US/splunkd/_raw/servicesNs/nobody/search/search/v2/jobs/rt_md_1761382392.309	81	200
2025-10-25 14:23:13.487	admin	GET	/en-US/splunkd/_raw/servicesNs/nobody/search/search/v2/jobs/rt_md_1761382392.309	81	200
2025-10-25 14:23:13.361	-	GET	/en-US/splunkd/_raw/servicesNs/admin/search/config/conf-searchconf	67	200
2025-10-25 14:23:13.184	admin	GET	/en-US/splunkd/_raw/servicesNs/admin/search/config/conf-searchconf	67	200
2025-10-25 14:23:13.187	-	GET	/en-US/splunkd/_raw/servicesNs/admin/search/static/appIcon.png	63	200
2025-10-25 14:23:13.042	-	GET	/en-US/splunkd/_raw/servicesNs/nobody/search/search/v2/jobs/rt_md_1761382392.309	81	200
2025-10-25 14:23:13.011	admin	GET	/en-US/splunkd/_raw/servicesNs/nobody/search/search/v2/jobs/rt_md_1761382392.309	81	200
2025-10-25 14:23:12.984	admin	GET	/en-US/splunkd/_raw/servicesNs/admin/search/search/v2/jobs/rt_md_1761382392.309	88	200
2025-10-25 14:23:12.986	admin	GET	/en-US/splunkd/_raw/servicesNs/admin/search/static/appIcon.png	63	200
2025-10-25 14:23:12.772	admin	GET	/en-US/splunkd/_raw/servicesNs/admin/search/static/appIcon.png	63	404
2025-10-25 14:23:12.648	admin	GET	/en-US/splunkd/_raw/servicesNs/nobody/splunk_instrumentation/admin/telemetry/general	85	200
2025-10-25 14:23:12.643	admin	GET	/en-US/splunkd/_raw/servicesNs/admin/search/apps/local/search	62	200
2025-10-25 14:23:12.596	admin	GET	/en-US/splunkd/_raw/servicesNs/admin/search/data/ui/visualizations	67	200

Step 7: Combined Summary Search

Built a single unified search that combines labeling, normalization, and metrics:

```
index=_internal sourcetype=splunkd_ui_access
| eval user_norm=lower(coalesce(user,user_name,"-unknown-"))
| eval class=case(status>=200 AND status<300,"Success",
                  status=404,"Not Found",
                  status>=500,"Server Error",
                  true(),"Other")
| eval url_len=len(uri_path)
| stats count,
  dc(uri_path) as unique_paths,
  avg(url_len) as avg_url_len
  by user_norm class
| sort - count
```

The screenshot shows the Splunk Enterprise search interface. At the top, there's a navigation bar with 'splunk-enterprise' and various tabs like 'Messages', 'Settings', 'Activity', 'Help', 'Find', and a search bar. Below the navigation is a 'New Search' section with a code editor containing a complex search command. The command uses `index=_internal sourcetype=splunkd_ui_access` and includes multiple `eval` and `if` statements to handle user identity normalization and HTTP status codes. Below the code editor is a summary bar indicating '369 events (10/24/25 2:30:00.000 PM to 10/25/25 2:30:01.000 PM)' and 'No Event Sampling'. Underneath is a table titled 'Statistics (S)' showing event counts by user_norm and class. The table has columns for 'user_norm', 'class', 'count', 'unique_paths', and 'avg_url_len'. The data shows admin users in Success, Other, and Not Found categories. The bottom part of the interface shows a histogram visualization and a detailed table of the same data.

user_norm	class	count	unique_paths	avg_url_len
admin	Success	389	96	66.44593818770227
-	Success	47	28	62.48425531914894
admin	Other	8	6	52.875
admin	Not Found	3	3	61.33333333333336
-	Other	2	2	4

Step 8: Save as a Report

Saved the final combined query as a reusable report for future dashboarding or alerting.

- Report name:**
Day56: UI Access – User/Class Summary
- Location:** Current App (*Search & Reporting*)

The screenshot shows the 'Day56: UI Access – User/Class Summary' report in the Splunk interface. It features a histogram visualization at the top with a legend for 'class', 'unique_paths', and 'avg_url_len'. Below the visualization is a table with the same data as the previous screenshot, showing event counts for different user_norm and class combinations. The table includes columns for 'user_norm', 'class', 'count', 'unique_paths', and 'avg_url_len'.

user_norm	class	count	unique_paths	avg_url_len
admin	Success	761	147	67.48735873858197
-	Success	67	34	64.98587462686567
admin	Other	22	15	52.09090909090909
admin	Not Found	19	6	63.8421052631579
-	Other	2	2	4

Reflection

- Where did coalesce() help?**
It unified user identity fields across different event types, avoiding missing or duplicated user counts when some logs used user and others user_name.
- if() vs case():**
Use if() for simple two-condition logic; use case() when classifying multiple conditions (e.g., HTTP status categories).
- Impact of lower()/upper():**
Normalizing case ensured accurate grouping. Without it, entries like *Admin*, *ADMIN*, and *admin* would be treated as separate users, skewing counts.

Summary

In this lab, I used **eval-based functions** to enhance Splunk search output by:

- Adding context and labels with if() and case()
- Cleaning inconsistent fields using coalesce()
- Standardizing text with lower() and upper()
- Measuring field lengths with len()
- Producing compact, accurate summaries ready for reports and dashboards

These SPL helpers form the backbone of efficient event normalization, data enrichment, and presentation within Splunk.