**Exercise - 1**

We'll be working with a log file named **syslog.log**, which contains logs related to *ticky*.

You can view this file using:

cat syslog.log

Copied!

content\_copy

The log lines follow a pattern similar to the ones we've seen before. Something like this:

May 27 11:45:40 ubuntu.local ticky: INFO: Created ticket [#1234] (username)

Jun 1 11:06:48 ubuntu.local ticky: ERROR: Connection to DB failed (username)

When the service runs correctly, it logs an INFO message to syslog. It then states what it did and states the username and ticket number related to the event. If the service encounters a problem, it logs an ERROR message to syslog. This error message indicates what was wrong and states the username that triggered the action that caused the problem.

In this section, we'll search and view different types of error messages. The error messages for *ticky* details the problems with the file with a timestamp for when each problem occurred.

These are a few kinds of listed error:

* Timeout while retrieving information
* The ticket was modified while updating
* Connection to DB failed
* Tried to add information to a closed ticket
* Permission denied while closing ticket
* Ticket doesn't exist

To grep all the logs from *ticky*, use the following command:

grep ticky syslog.log

Copied!

content\_copy

Output:

Text

Description automatically generated

In order to search all the **ERROR** logs, use the following command:

grep "ERROR" syslog.log

Copied!

content\_copy

Output:

Text

Description automatically generated

To enlist all the ERROR messages of specific kind use the below syntax.

**Syntax:** grep ERROR [message] [file-name]

grep "ERROR Tried to add information to closed ticket" syslog.log

Copied!

content\_copy

Output:

A picture containing text, window, file

Description automatically generated

Let's now write a few regular expressions using a python3 interpreter.

We can also grep the ERROR/INFO messages in a pythonic way using a regular expression. Let's now write a few regular expressions using a **python3** interpreter.

Open **Python shell** using the command below:

python3

Copied!

content\_copy

This opens a **Shell**. Python provides a **Python Shell** (also known as **Python** Interactive **Shell**), which is used to execute a single **Python** command and get the result.

Import the regular expression module (re).

import re  
line = "May 27 11:45:40 ubuntu.local ticky: INFO: Created ticket [#1234] (username)"

Copied!

content\_copy

To match a string stored in **line** variable, we use the search() method by defining a pattern.

re.search(r"ticky: INFO: ([\w ]\*) ", line)

Copied!

content\_copy

Output:

<\_sre.SRE\_Match object; span=(29, 57), match='ticky: INFO: Created ticket '>

Copied!

content\_copy

You can also get the **ERROR** message as we did for the **INFO** log above from the ERROR log line.

line = "May 27 11:45:40 ubuntu.local ticky: ERROR: Error creating ticket [#1234] (username)"

Copied!

content\_copy

To match a string stored in a **line** variable, we use the search() method by defining a pattern.

re.search(r"ticky: ERROR: ([\w ]\*) ", line)

Copied!

content\_copy

Output:

<\_sre.SRE\_Match object; span=(29, 65), match='ticky: ERROR: Error creating ticket '>

Copied!

content\_copy

Now that you know how to use regular expressions with Python, start fetching logs of *ticky* for a specific username. We'll need them in later sections.

**Exercise - 2**

Now, use the Python interactive shell to create a dictionary.

fruit = {"oranges": 3, "apples": 5, "bananas": 7, "pears": 2}

Copied!

content\_copy

Call the sorted function to sort the items in the dictionary.

sorted(fruit.items())

Copied!

content\_copy

Output:

[('apples', 5), ('bananas', 7), ('oranges', 3), ('pears', 2)]

Copied!

content\_copy

We'll now sort the dictionary using the item's key. For this use the **operator** module.

Pass the function itemgetter() as an argument to the sorted() function. Since the second element of tuple needs to be sorted, pass the argument 0 to the itemgetter function of the **operator** module.

import operator

Copied!

content\_copy

sorted(fruit.items(), key=operator.itemgetter(0))

Copied!

content\_copy

Output:

[('apples', 5), ('bananas', 7), ('oranges', 3), ('pears', 2)]

Copied!

content\_copy

To sort a dictionary based on its **values**, pass the argument 1 to the itemgetter function of the **operator** module.

sorted(fruit.items(), key=operator.itemgetter(1))

Copied!

content\_copy

Output:

[('pears', 2), ('oranges', 3), ('apples', 5), ('bananas', 7)]

Copied!

content\_copy

Finally, you can also reverse the order of the sort using the **reverse** parameter. This parameter takes in a boolean argument.

To sort the fruit object from most to least occurrence, we pass the argument reverse=True.

sorted(fruit.items(), key = operator.itemgetter(1), reverse=True)

Copied!

content\_copy

Output:

[('bananas', 7), ('apples', 5), ('oranges', 3), ('pears', 2)]

Copied!

content\_copy

You can see the fruit object is now sorted from the most to the least number of occurrences.

Great job practice these skills! You can further practice this by sorting the logs that you would fetch using regular expressions from the previous section.

Exit the shell using exit().

**Exercise - 3**

We'll now work with a file named **csv\_to\_html.py**. This file converts the data in a CSV file into an HTML file that contains a table with the data. Let's practice this with an example file.

Create a new CSV file.

nano user\_emails.csv

Copied!

content\_copy

Add the following data into the file:

Full Name, Email Address  
Blossom Gill, blossom@abc.edu  
Hayes Delgado, nonummy@utnisia.com  
Petra Jones, ac@abc.edu  
Oleg Noel, noel@liberomauris.ca  
Ahmed Miller, ahmed.miller@nequenonquam.co.uk  
Macaulay Douglas, mdouglas@abc.edu  
Aurora Grant, enim.non@abc.edu  
Madison Mcintosh, mcintosh@nisiaenean.net  
Montana Powell, montanap@semmagna.org  
Rogan Robinson, rr.robinson@abc.edu  
Simon Rivera, sri@abc.edu  
Benedict Pacheco, bpacheco@abc.edu  
Maisie Hendrix, mai.hendrix@abc.edu  
Xaviera Gould, xlg@utnisia.net  
Oren Rollins, oren@semmagna.com  
Flavia Santiago, flavia@utnisia.net  
Jackson Owens, jackowens@abc.edu  
Britanni Humphrey, britanni@ut.net  
Kirk Nixon, kirknixon@abc.edu  
Bree Campbell, breee@utnisia.net

Copied!

content\_copy

Save the file by clicking Ctrl-o, Enter key, and Ctrl-x.

Give executable permission to the script file **csv\_to\_html.py**.

sudo chmod +x csv\_to\_html.py

Copied!

content\_copy

To visualize the data in the **user\_emails.csv** file, you have to generate a webpage that'll be served by the webserver running on the machine.

The script **csv\_to\_html.py** takes in two arguments, the CSV file, and location that would host the HTML page generated. Give write permission to the directory that would host that HTML page:

sudo chmod o+w /var/www/html

Copied!

content\_copy

Next, run the script **csv\_to\_html.py** script by passing two arguments: user\_emails.csv file and the path /var/www/html/. Also, append a name to the path with an HTML extension. This should be the name that you want the HTML file to be created with.

./csv\_to\_html.py user\_emails.csv /var/www/html/<html-filename>.html

Copied!

content\_copy

Replace <html-filename> with the new name.

Navigate to the /var/www/html directory. Here, you'll find an HTML file created with the filename you passed to the above script.

ls /var/www/html

Copied!

content\_copy

Now, to view this HTML page, open any web-browser and enter the following URL in the search bar.

**[linux-instance-external-IP]**/**[html-filename].html**

Output:

Table

Description automatically generated

You should now be able to visualize the data within the user\_emails.csv file on a webpage.

**Generate reports**

Now, we're going to practice creating a script, named **ticky\_check.py**, that generates two different reports from this internal ticketing system log file i.e., syslog.log. This script will create the following reports:

* **The ranking of errors generated by the system**: A list of all the error messages logged and how many times each error was found, sorted by the most common error to the least common error. This report doesn't take into account the users involved.
* **The user usage statistics for the service**: A list of all users that have used the system, including how many info messages and how many error messages they've generated. This report is sorted by username.

To create these reports write a python script named ticky\_check.py. Use nano editor for this.

nano ticky\_check.py

Copied!

content\_copy

Add the shebang line.

#!/usr/bin/env python3

Copied!

content\_copy

Here's your challenge: Write a script to generate two different reports based on the ranking of errors generated by the system and the user usage statistics for the service. You'll write the script on your own, but we'll guide you throughout.

First, import all the Python modules that you'll use in this Python script. After importing the necessary modules, initialize two dictionaries: one for the number of different error messages and another to count the number of entries for each user (splitting between INFO and ERROR).

Now, parse through each log entry in the syslog.log file by iterating over the file.

For each log entry, you'll have to first check if it matches the **INFO** or **ERROR** message formats. You should use regular expressions for this. When you get a successful match, add one to the corresponding value in the per\_user dictionary. If you get an **ERROR** message, add one to the corresponding entry in the error dictionary by using proper data structure.

After you've processed the log entries from the syslog.log file, you need to sort both the per\_user and error dictionary before creating CSV report files.

Keep in mind that:

* The error dictionary should be sorted by the number of errors from most common to least common.
* The user dictionary should be sorted by username.

Insert column names as ("Error", "Count") at the zero index position of the sorted error dictionary. And insert column names as ("Username", "INFO", "ERROR") at the zero index position of the sorted per\_user dictionary.

After sorting these dictionaries, store them in two different files: **error\_message.csv** and **user\_statistics.csv**.

Save the **ticky\_check.py** file by clicking Ctrl-o, Enter key, and Ctrl-x.

**Visualize reports**

First, give executable permission to the Python script ticky\_check.py.

chmod +x ticky\_check.py

Copied!

content\_copy

Run the ticky\_check.py by using the following command:

./ticky\_check.py

Copied!

content\_copy

Executing ticky\_check.py will generate two report file \_\_error\_message.csv \_\_and **user\_statistics.csv**.

You can now visualize the \_\_error\_message.csv \_\_and **user\_statistics.csv** by converting them to HTML pages. To do this, pass the files one by one to the script **csv\_to \_html.py** file, like we did in the previous section.

To convert the **error\_message.csv** into HTML file run the following command:

./csv\_to\_html.py error\_message.csv /var/www/html/<html-filename>.html

Copied!

content\_copy

Replace <html-filename> with the name of your choice.

To convert **user\_statistics.csv** into HTML file, run the following command:

./csv\_to\_html.py user\_statistics.csv /var/www/html/<html-filename>.html

Copied!

content\_copy

Replace <html-filename> with the new name

Now, to view these HTML pages, open any web-browser and enter the following URL in the search bar.

**[linux-instance-external-IP]**/**[html-filename].html**

Output:

Table

Description automatically generated

Table

Description automatically generated

Click *Check my progress* to verify the objective.

Generate two CSV report files and visualize data on the web

Check my progress

*Successfully visualized CSV data on HTML table.*

**Congratulations!**