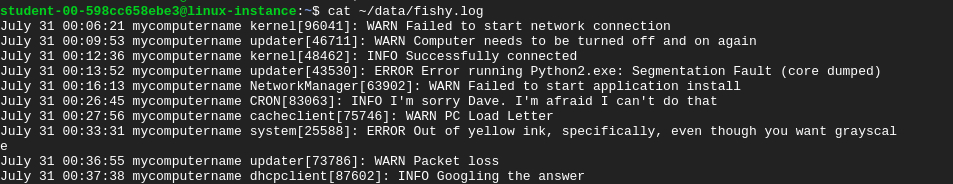
**View log file**

In the /data directory, there's a file named fishy.log, which contains the system log. Log entries are written in this format:

Month Day hour:minute:second mycomputername "process\_name"["random 5 digit number"] "ERROR/INFO/WARN" "Error description"

For every process, the runtime log that's generated contains a timestamp and appropriate message alongside. You can view all logs using the command below:

cat ~/data/fishy.log

Output: 

**Find an error**

In this lab, we'll search for the CRON error that failed to start. To do this, we'll use a python script to search log files for a particular type of ERROR log. In this case, we'll search for a CRON error within the fishy.log file that failed to start by narrowing our search to "CRON ERROR Failed to start".

To get started, let's create a python script named **find\_error.py** within scripts directory using nano editor.

cd ~/scripts

nano find\_error.py

Add the shebang line:

#!/usr/bin/env python3

Import the necessary Python modules:

import sys

import os

import re

The **sys** module provides information about the Python interpreter's constants, functions, and methods. The **os** module provides a portable way of using operating system dependent functionality with Python.

Regular Expression (RegEx) is a sequence of characters that defines a search pattern. We can use regular expressions using **re** module.

Now, write a function error\_search that takes log\_file as a parameter and returns returned\_errors. Define the error\_search function and pass the log file to it as a parameter.

def error\_search(log\_file):

To allow us to search all log files for any type of logs, we'll be making our script consistent and dynamic.

Define an input function to receive the type of ERROR that the end-user would like to search and assign to a variable named error.

The input() function takes the input from the user and then evaluates the expression. This means Python automatically identifies whether the user entered a string, a number, or a list. If the input provided isn't correct then Python will raise either a syntax error or exception. The program flow will stop until the user has given an input.

Later in the script, we'll iterate over this user input and the log file to produce results. Following the input function, now initialize the list returned\_errors. This will enlist all the ERROR logs as specified by the end-user through the input function.

error = input("What is the error? ")

returned\_errors = []

Use the Python file's handling methods to open the log file in reading mode and use 'UTF-8' encoding.

with open(log\_file, mode='r',encoding='UTF-8') as file:

We'll now read each log separately from the fishy.log file using the readlines() method. As mentioned earlier, we'll iterate over user input to get the desired search results. For this, we'll create a list to store all the patterns (user input) that will be searched. This list is named error\_patterns and, initially it has a pattern "**error**" to filter out all the ERROR logs only. You can change this to view other types of logs such as INFO and WARN. You can also empty initialize the list to fetch all types of logs, irrespective of their type.

We'll add the whole user input to this list error\_patterns.

for log in file.readlines():

error\_patterns = ["error"]

for i in range(len(error.split(' '))):

error\_patterns.append(r"{}".format(error.split(' ')[i].lower()))

Now, let's use the search() method (present in re module) to check whether the file fishy.log has the user defined pattern and, if it is available, append them to the list returned\_errors.

if all(re.search(error\_pattern, log.lower()) for error\_pattern in error\_patterns):

returned\_errors.append(log)

Next, close the file fishy.log and return the results stored in the list returned\_errors.

file.close()

return returned\_errors

Great job! You've successfully defined a function to store all the logs defined as a CRON error that fails to start. In the next section, we'll generate a new file consisting of the logs based on your search within /data directory.

**Create an output file**

Let's define another function file\_output that takes returned\_errors, returned by a previous function, as a formal parameter.

def file\_output(returned\_errors):

Using Python file handling methods, write returned\_errors into the errors\_found.log file by opening the file in writing mode. For defining the output file, we'll use the method os.path.expanduser ('~'), which returns the home directory of your system instance. Then, we'll concatenate this path (to the home directory) to the file errors\_found.log in /data directory.

with open(os.path.expanduser('~') + '/data/errors\_found.log', 'w') as file:

Next, write all the logs to the output file by iterating over returned\_errors.

for error in returned\_errors:

file.write(error)

And finally, close the file.

file.close()

**Function call**

Now, let's call the functions and run the script.

Define the main function and call both functions that we defined in the earlier sections.

The variable log\_file takes in the path to the log file passed as a parameter. In our case, the file is fishy.log. Call the first function i.e., error\_search() and pass the variable log\_file to the function. This function will search and return a list of errors that would be stored in the variable returned\_errors. Call the second function file\_output and pass the variable returned\_errors as a parameter.

sys.exit(0) is used to exit from Python, the optional argument passed can be an integer giving the exit status (defaulting to zero), or another type of object. If it is an integer, zero is considered "successful termination" and any nonzero value is considered an "abnormal termination" by shells.

if \_\_name\_\_ == "\_\_main\_\_":

log\_file = sys.argv[1]

returned\_errors = error\_search(log\_file)

file\_output(returned\_errors)

sys.exit(0)

The complete file find\_error.py should now look like this:

#!/usr/bin/env python3

import sys

import os

import re

def error\_search(log\_file):

error = input("What is the error? ")

returned\_errors = []

with open(log\_file, mode='r',encoding='UTF-8') as file:

for log in file.readlines():

error\_patterns = ["error"]

for i in range(len(error.split(' '))):

error\_patterns.append(r"{}".format(error.split(' ')[i].lower()))

if all(re.search(error\_pattern, log.lower()) for error\_pattern in error\_patterns):

returned\_errors.append(log)

file.close()

return returned\_errors

def file\_output(returned\_errors):

with open(os.path.expanduser('~') + '/data/errors\_found.log', 'w') as file:

for error in returned\_errors:

file.write(error)

file.close()

if \_\_name\_\_ == "\_\_main\_\_":

log\_file = sys.argv[1]

returned\_errors = error\_search(log\_file)

file\_output(returned\_errors)

sys.exit(0)

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

Make the file executable before running it.

sudo chmod +x find\_error.py

Now, run the file by passing the path to fishy.log as a parameter to the script.

./find\_error.py ~/data/fishy.log

This script will now prompt for the type of error to be searched. Continue by entering the following type of error:

CRON ERROR Failed to start

On successful execution, this will generate an **errors\_found.log** file, where you will find all the ERROR logs based on your search. You can view the ERROR log using the command below:

cat ~/data/errors\_found.log

This will output the following: fa35b5be265b9a7b.png

Click *Check my progress* to verify the objective.

Generate a file consisting of required error

Check my progress

**Congratulations!**

Congrats! You've written a script to search the log file for the exact error, and then output that error into a separate file for further analysis. As an IT specialist, this tool will be super helpful, allowing you to use Python scripting to filter out and analyze all types of logs.