

Working with Log Files | Qwiklabs

Qwiklabs

13-16 minutes

Introduction

Imagine one of your colleagues is struggling with a program that keeps throwing an error. Unfortunately, the program's source code is too complicated to easily find the error there. The good news is that the program outputs a log file you can read! Let's write a script to search the log file for the exact error, then output that error into a separate file so you can work out what's wrong.

What you'll do

- Write a script to search the log file using regex to find for the exact error.
- Report the error into a separate file so you know what's wrong for further analysis.

You'll have 90 minutes to complete this lab.

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:

```
student-00-598cc658ebe3@linux-instance:~$ cat ~/data/fishy.log
July 31 00:06:21 mycomputername kernel[96041]: WARN Failed to start network connection
July 31 00:09:53 mycomputername updater[46711]: WARN Computer needs to be turned off and on again
July 31 00:12:36 mycomputername kernel[48462]: INFO Successfully connected
July 31 00:13:52 mycomputername updater[43530]: ERROR Error running Python2.exe: Segmentation Fault (core dumped)
July 31 00:16:13 mycomputername NetworkManager[63902]: WARN Failed to start application install
July 31 00:26:45 mycomputername CRON[83063]: INFO I'm sorry Dave. I'm afraid I can't do that
July 31 00:27:56 mycomputername cacheclient[75746]: WARN PC Load Letter
July 31 00:33:31 mycomputername system[25588]: ERROR Out of yellow ink, specifically, even though you want grayscale
July 31 00:36:55 mycomputername updater[73786]: WARN Packet loss
July 31 00:37:38 mycomputername dhcpclient[87602]: INFO Googling the answer
```

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:

```

student-00-598cc658ebe3@linux-instance:~/scripts$ cat ../data/error_found.log
July 31 04:11:32 mycomputername CRON[51253]: ERROR: Failed to start CRON job due to script syntax error. Inform the CRON job owner!

```