

# Log Analysis Using Regular Expressions

1 hour 30 minutesFree

Rate Lab

## Introduction

Imagine your company uses a server that runs a service called *ticky*, an internal ticketing system. The service logs events to syslog, both when it runs successfully and when it encounters errors.

The service's developers need your help getting some information from those logs so that they can better understand how their software is used and how to improve it. So, for this lab, you'll write some automation scripts that will process the system log and generate reports based on the information extracted from the log files.

### What you'll do

- Use regex to parse a log file
  - Append and modify values in a dictionary
  - Write to a file in CSV format
  - Move files to the appropriate directory for use with the CSV->HTML converter
- You'll have 90 minutes to complete this lab.

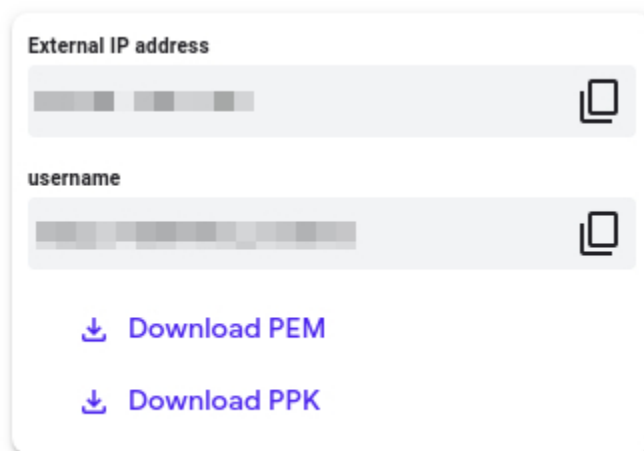
## Start the lab

You'll need to start the lab before you can access the materials in the virtual machine OS. To do this, click the green “Start Lab” button at the top of the screen.

**Note:** For this lab you are going to access the **Linux VM** through your **local SSH Client**, and not use the **Google Console** (**Open GCP Console** button is not available for this lab).

A green rectangular button with the text "Start Lab" in white.

After you click the “Start Lab” button, you will see all the SSH connection details on the left-hand side of your screen. You should have a screen that looks like this:

A light gray rounded rectangle containing SSH connection details. It has two input fields: "External IP address" and "username", each with a copy icon to its right. Below these are two links: "Download PEM" and "Download PPK", each preceded by a download icon.

External IP address

username

Download PEM

Download PPK

## Accessing the virtual machine

Please find one of the three relevant options below based on your device's operating system.

**Note:** Working with Qwiklabs may be similar to the work you'd perform as an **IT Support Specialist**; you'll be interfacing with a cutting-edge technology that requires multiple steps to access, and perhaps healthy doses of patience and persistence(!). You'll also be using **SSH** to enter the labs -- a critical skill in IT Support that you'll be able to practice through the labs.

## Option 1: Windows Users: Connecting to your VM

In this section, you will use the PuTTY Secure Shell (SSH) client and your VM's External IP address to connect.

### Download your PPK key file

You can download the VM's private key file in the PuTTY-compatible **PPK** format from the Qwiklabs Start Lab page. Click on **Download PPK**.

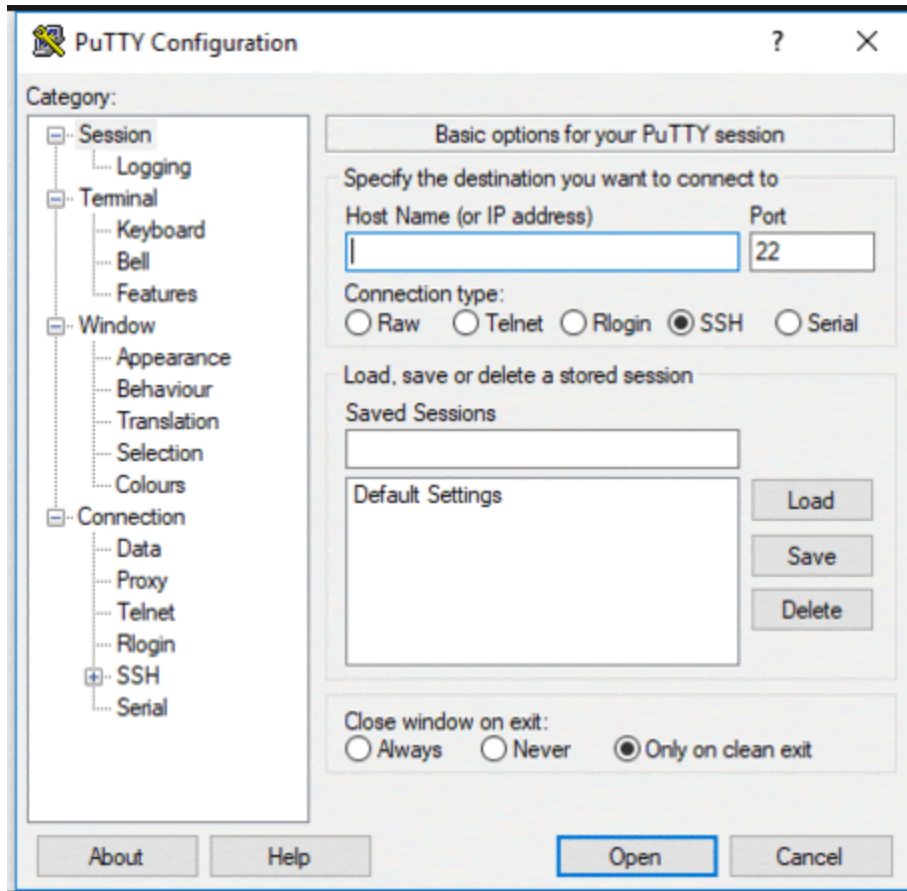
 [Download PEM](#)

 [Download PPK](#) 

### Connect to your VM using SSH and PuTTY

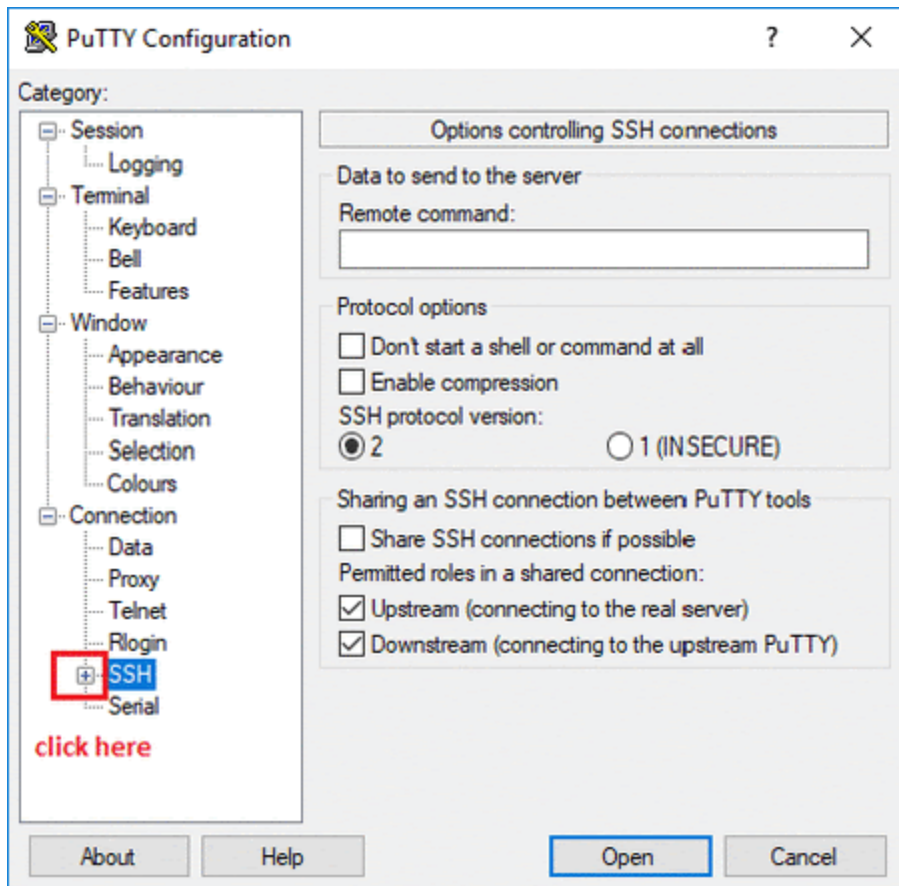
1. You can download Putty from [here](#)
2. In the **Host Name (or IP address)** box, enter  
username@external\_ip\_address.

**Note:** Replace **username** and **external\_ip\_address** with values provided in the lab.



3. In the **Category** list, expand **SSH**.
4. Click **Auth** (don't expand it).
5. In the **Private key file for authentication** box, browse to the PPK file that you downloaded and double-click it.
6. Click on the **Open** button.

**Note:** PPK file is to be imported into PuTTY tool using the Browse option available in it. It should not be opened directly but only to be used in PuTTY.



7. Click **Yes** when prompted to allow a first connection to this remote SSH server. Because you are using a key pair for authentication, you will not be prompted for a password.

### Common issues

If PuTTY fails to connect to your Linux VM, verify that:

- You entered **<username>@<external ip address>** in PuTTY.
- You downloaded the fresh new PPK file for this lab from Qwiklabs.
- You are using the downloaded PPK file in PuTTY.

## Option 2: OSX and Linux users: Connecting to your VM via SSH

**Download your VM's private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



### Connect to the VM using the local Terminal application

A **terminal** is a program which provides a **text-based interface for typing commands**. Here you will use your terminal as an SSH client to connect with lab provided Linux VM.

1. Open the Terminal application.
  - To open the terminal in Linux use the shortcut key **Ctrl+Alt+t**.
  - To open terminal in **Mac (OSX)** enter **cmd + space** and search for **terminal**.
2. Enter the following commands.

**Note:** Substitute the **path/filename for the PEM** file you downloaded, **username** and **External IP Address**.

You will most likely find the PEM file in **Downloads**. If you have not changed the download settings of your system, then the path of the PEM key will be **~/Downloads/qwikLABS-XXXXXX.pem**

```
chmod 600 ~/Downloads/qwikLABS-XXXXXX.pem
ssh -i ~/Downloads/qwikLABS-XXXXXX.pem username@External Ip Address

gcpstagingeduit1370_student@linux-instance:~$ ssh -i ~/Downloads/qwikLABS-L923-42090.pem gcpstagingeduit1370_student@35.239.106.192
The authenticity of host '35.239.106.192 (35.239.106.192)' can't be established.
ECDSA key fingerprint is SHA256:vrz8b4aYUtruFh0A6wZn6OzyloqqPEfh931olvxITm8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '35.239.106.192' (ECDSA) to the list of known hosts.
Linux linux-instance 4.9.0-9-amd64 #1 SMP Debian 4.9.168-1+deb9u2 (2019-05-13) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
gcpstagingeduit1370_student@linux-instance:~$
```

### Option 3: Chrome OS users: Connecting to your VM via SSH

**Note:** Make sure you are not in **Incognito/Private mode** while launching the application.  
**Download your VM's private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



## Connect to your VM

1. Add Secure Shell from [here](#) to your Chrome browser.
2. Open the Secure Shell app and click on **[New Connection]**.

This is a screenshot of the 'New Connection' dialog box in the Secure Shell application. The dialog has a title bar that says '[New Connection]'. Below the title bar is a text field containing 'username@hostname or free form text'. Underneath this are three input fields: 'username', 'hostname', and 'port'. Below these is a field for 'SSH relay server options'. Further down is an 'Identity' dropdown menu set to '[default]' with an 'Import...' button next to it. Below that is an 'SSH Arguments' field with the placeholder text 'extra command line arguments'. Then is a 'Current profile' field set to 'default'. At the bottom is a 'Mount Path' field with the text 'the default path is the user's home directory'. At the very bottom of the dialog are several buttons: '[DEL] Delete', 'Options', 'SFTP Mount', 'SFTP', and '[ENTER] Connect'. A red arrow points from the left towards the '[New Connection]' title bar.

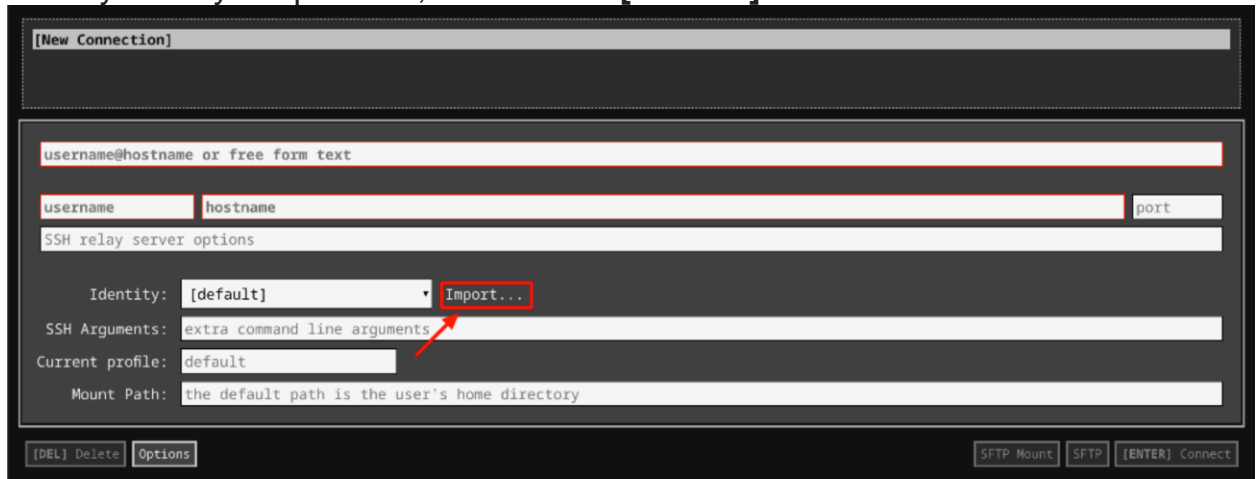
3. In the **username** section, enter the username given in the Connection Details Panel of the lab. And for the **hostname** section, enter the external IP of your VM instance that is mentioned in the Connection Details Panel of the lab.

This is another screenshot of the 'New Connection' dialog box, similar to the one above. In this version, the 'username' and 'hostname' input fields are highlighted with red rectangular boxes. A red arrow points from the left towards the 'username' field.

4. In the **Identity** section, import the downloaded PEM key by clicking on the **Import...** button beside the field. Choose your PEM key and click on the **OPEN** button.

**Note:** If the key is still not available after importing it, refresh the application, and select it from the **Identity** drop-down menu.

5. Once your key is uploaded, click on the **[ENTER] Connect** button below.



6. For any prompts, type **yes** to continue.
7. You have now successfully connected to your Linux VM.

You're now ready to continue with the lab!

## 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
```

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
```

Output:

```
student-04-e7f6cdb39f01@linux-instance:~$ grep "ticky" syslog.log
Jan 31 00:09:39 ubuntu.local ticky: INFO Created ticket [#4217] (mdouglas)
Jan 31 00:16:25 ubuntu.local ticky: INFO Closed ticket [#1754] (noel)
Jan 31 00:21:30 ubuntu.local ticky: ERROR The ticket was modified while updating (breee)
Jan 31 00:44:34 ubuntu.local ticky: ERROR Permission denied while closing ticket (ac)
Jan 31 01:00:50 ubuntu.local ticky: INFO Commented on ticket [#4709] (blossom)
Jan 31 01:29:16 ubuntu.local ticky: INFO Commented on ticket [#6518] (rr.robinson)
Jan 31 01:33:12 ubuntu.local ticky: ERROR Tried to add information to closed ticket (mcintosh)
Jan 31 01:43:10 ubuntu.local ticky: ERROR Tried to add information to closed ticket (jackowens)
Jan 31 01:49:29 ubuntu.local ticky: ERROR Tried to add information to closed ticket (mdouglas)
Jan 31 02:30:04 ubuntu.local ticky: ERROR Timeout while retrieving information (oren)
```

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

```
grep "ERROR" syslog.log
```

Output:

```
student-04-e7f6cdb39f01@linux-instance:~$ grep "ERROR" syslog.log
Jan 31 00:21:30 ubuntu.local ticky: ERROR The ticket was modified while updating (breee)
Jan 31 00:44:34 ubuntu.local ticky: ERROR Permission denied while closing ticket (ac)
Jan 31 01:33:12 ubuntu.local ticky: ERROR Tried to add information to closed ticket (mcintosh)
Jan 31 01:43:10 ubuntu.local ticky: ERROR Tried to add information to closed ticket (jackowens)
Jan 31 01:49:29 ubuntu.local ticky: ERROR Tried to add information to closed ticket (mdouglas)
Jan 31 02:30:04 ubuntu.local ticky: ERROR Timeout while retrieving information (oren)
Jan 31 02:55:31 ubuntu.local ticky: ERROR Ticket doesn't exist (xlg)
Jan 31 03:05:35 ubuntu.local ticky: ERROR Timeout while retrieving information (ahmed.miller)
Jan 31 03:08:55 ubuntu.local ticky: ERROR Ticket doesn't exist (blossom)
Jan 31 03:39:27 ubuntu.local ticky: ERROR The ticket was modified while updating (bpacheco)
```

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
```

Output:

```
student-04-e7f6cdb39f01@linux-instance:~$ grep "ERROR Tried to add information to closed ticket" syslog.log
Jan 31 01:33:12 ubuntu.local ticky: ERROR Tried to add information to closed ticket (mcintosh)
Jan 31 01:43:10 ubuntu.local ticky: ERROR Tried to add information to closed ticket (jackowens)
Jan 31 01:49:29 ubuntu.local ticky: ERROR Tried to add information to closed ticket (mdouglas)
Jan 31 04:31:49 ubuntu.local ticky: ERROR Tried to add information to closed ticket (oren)
Jan 31 05:12:39 ubuntu.local ticky: ERROR Tried to add information to closed ticket (oren)
Jan 31 05:18:45 ubuntu.local ticky: ERROR Tried to add information to closed ticket (sri)
Jan 31 08:01:40 ubuntu.local ticky: ERROR Tried to add information to closed ticket (jackowens)
Jan 31 09:04:27 ubuntu.local ticky: ERROR Tried to add information to closed ticket (noel)
Jan 31 11:04:02 ubuntu.local ticky: ERROR Tried to add information to closed ticket (breere)
Jan 31 11:58:33 ubuntu.local ticky: ERROR Tried to add information to closed ticket (ahmed.miller)
Jan 31 12:58:16 ubuntu.local ticky: ERROR Tried to add information to closed ticket (nonummy)
Jan 31 14:56:35 ubuntu.local ticky: ERROR Tried to add information to closed ticket (noel)
```

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
```

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)"
```

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

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

Output:

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

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)"
```

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)
```

Output:

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

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}
```

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

```
sorted(fruit.items())
```

Output:

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

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

Output:

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

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))
```

Output:

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

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)
```

Output:

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

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
```

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
Macauley Douglas, mdouglas@abc.edu
Aurora Grant, enim.non@abc.edu
Madison Mcintosh, mcintosh@nisiaenean.net
Montana Powell, montanap@sem magna.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@sem magna.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
```

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
```

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
```

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
```

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
```

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:

## User Emails

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

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
```

Add the shebang line.

```
#!/usr/bin/env python3
```

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
```

Run the `ticky_check.py` by using the following command:

```
./ticky_check.py
```

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
```

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
```

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:



## Error Message

Error	Count
Timeout while retrieving information	15
Connection to DB failed	13
Tried to add information to closed ticket	12
Permission denied while closing ticket	10
The ticket was modified while updating	9
Ticket doesn't exist	7

## User Statistics

Username	INFO	ERROR
ac	2	2
ahmed.miller	2	4
blossom	2	6
bpacheco	0	2
breee	1	5
britanni	1	1
enim.non	2	3
flavia	0	5

```

#!/usr/bin/env python3
import re
import sys
import operator
import csv

users={}
errors={}
user_count = {'INFO':0, 'ERROR':0}

f=open("syslog.log")
for line in f:
    line=line.strip()
    username=(re.search(r"\((.*)\)",line)).group(1)
    msg=(re.search(r"(ERROR|INFO)",line)).group(1)
    if (username not in users):
        users[username] =user_count
    users[username][msg]+=1

    if msg=="ERROR":
        err=(re.search(r"ERROR (.*) ",line)).group(1)
        if (err not in errors):
            errors[err]=0
        errors[err]+=1
f.close()

users2=[]
errors2=[]
for key in sorted(users.keys()):
    users2.append([key,users[key]["INFO"],users[key]["ERROR"]])

for key, value in sorted(errors.items(), key=lambda item: item[1],reverse=True):
    errors2.append([key, value])

users2.insert(0,["Username","INFO","ERROR"])
errors2.insert(0,["Error","Count"])

fe=open("error_message.csv","w")
fu=open("user_statistics.csv","w")

writer1=csv.writer(fe)
writer2=csv.writer(fu)
writer1.writerows(errors2)
writer2.writerows(users2)

fe.close()
fu.close()

```