Start Lab 01:30:00

Handling Files

1 hour 30 minutes Free ★★★★☆ Rate Lab

Introduction

For this lab, imagine you are an IT Specialist at a medium-sized company. The Human Resources Department at your company wants you to find out how many people are in each department. You need to write a Python script that reads a CSV file containing a list of the employees in the organization, counts how many people are in each department, and then generates a report using this information. The output of this script will be a plain text file. We will guide you through each step of the lab.

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).

for this lab).

Start Lab

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:

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

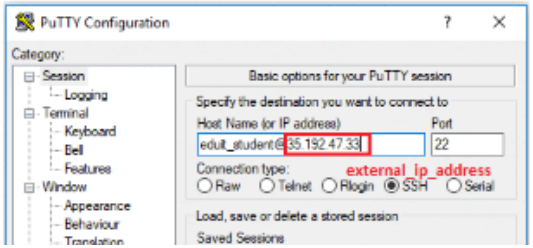
You can download the VM's private key file in the PuTTY-compatible **PPK** format from the Qwiklabs Start Lab page. Click on **Download PPK**.

Connect to your VM using SSH and PuTTY

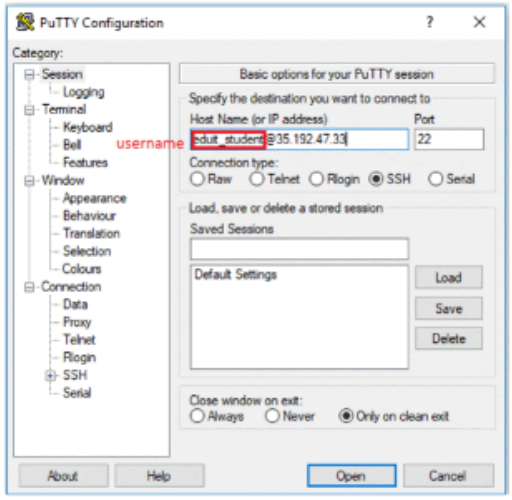
1. You can download Putty from here

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.



Note: Replace **username** and **external_ip_address** with values provided in the lab.



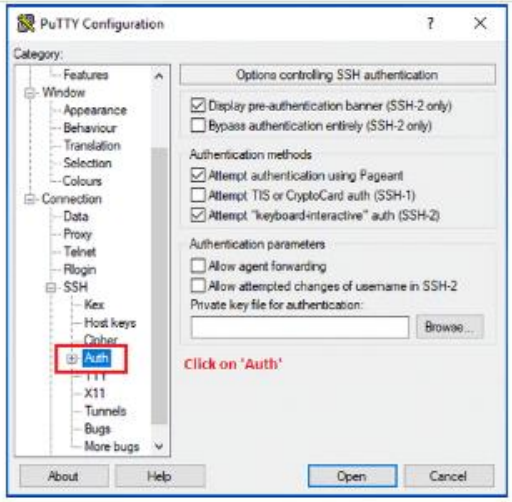
3. In the **Category** list, expand **SSH**.

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

- 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.

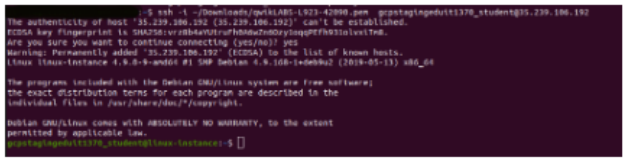
Note: Substitute the **path/filename for the PEM** file you downloaded, **username** and **External IP Address**.

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-**

settings of your system, then the path of the PEM key will be **~/Downloads/qwikLABS-XXXXX.pem**

```
chmod 600 ~/Downloads/qwikLABS-XXXXX.pem
```

```
ssh -i ~/Downloads/qwikLABS-XXXXX.pem username@External Ip Address
```



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

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]**.

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.

4. In the **Identity** section, import the downloaded PEM key by clicking on the **Import...**

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!

Prerequisites

We have created the employee list for you. Navigate to the data directory using the following command:

Prerequisites

We have created the employee list for you. Navigate to the data directory using the following command:

```
cd data
```

To find the data, list the files using the following command:

```
ls
```

You can now see a file called **employees.csv**, where you will find your data. You can also see a directory called scripts. We will write the python script in this directory.

To view the contents of the file, enter the following command:

```
cat employees.csv
```

Let's start by writing the script. You will write this python script in the scripts directory. Go to the scripts directory by using the following command:

```
cd ~/scripts
```

Create a file named **generate_report.py** using the following command:

```
nano generate_report.py
```

You will write your python script in this **generate_report.py** file. This script begins with a

You will write your python script in this **generate_report.py** file. This script begins with a line containing the #! character combination, which is commonly called hash bang or shebang, and continues with the path to the interpreter. If the kernel finds that the first two bytes are #! then it uses the rest of the line as an interpreter and passes the file as an argument. We will use the following shebang in this script:

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

Convert employee data to dictionary

The goal of the script is to read the CSV file and generate a report with the total number of people in each department. To achieve this, we will divide the script into three functions.

Let's start with the first function: *read_employees()*. This function receives a CSV file as a parameter and returns a list of dictionaries from that file. For this, we will use the CSV module.

The CSV module uses classes to read and write tabular data in CSV format. The CSV library allows us to both read from and write to CSV files.

Now, import the CSV module.

```
import csv
```

Define the function *read_employees*. This function takes file_location (path to employees.csv) as a parameter.

```
def read_employees(csv_file_location):
```

Open the CSV file by calling **open** and then **csv.DictReader**.

`DictReader` creates an object that operates like a regular reader (an object that iterates over lines in the given CSV file), but also maps the information it reads into a dictionary where keys are given by the optional *fieldnames* parameter. If we omit the *fieldnames* parameter, the values in the first row of the CSV file will be used as the keys. So, in this case, the first line of the CSV file has the keys and so there's no need to pass *fieldnames* as a parameter.

We also need to pass a dialect as a parameter to this function. There isn't a well-defined standard for comma-separated value files, so the parser needs to be flexible. Flexibility here means that there are many parameters to control how csv parses or writes data. Rather than passing each of these parameters to the reader and writer separately, we group them together conveniently into a dialect object.

Dialect classes can be registered by name so that callers of the CSV module don't need to know the parameter settings in advance. We will now register a dialect **empDialect**.

```
csv.register_dialect('empDialect', skipinitialspace=True, strict=True)
```

The main purpose of this dialect is to remove any leading spaces while parsing the CSV file.

The function will look similar to:

```
  employee_file = csv.DictReader(open(csv_file_location), dialect =
'empDialect')
```

You now need to iterate over the CSV file that you opened, i.e., employee_file. When you iterate over a CSV file, each iteration of the loop produces a dictionary from strings (key) to

iterate over a CSV file, each iteration of the loop produces a dictionary from strings (key) to strings (value).

Append the dictionaries to an empty initialised list **employee_list** as you iterate over the CSV file.

```
employee_list = []
for data in employee_file:
  employee_list.append(data)
```

Now return this list.

```
return employee_list
```

To test the function, call the function and save it to a variable called `employee_list.` Pass the path to employees.csv as a parameter to the function. Print the variable `employee_list` to check whether it returns a list of dictionaries.

```
employee_list = read_employees('<file_location>')
print(employee_list)
```

Replace `<file_location>` with the path to the employees.csv (this should look similar to the path */home/<username>/data/employees.csv*). Replace `<username>` with the one mentioned in Connection Details Panel at left hand side.

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

For the file to run it needs to have execute permission (x). Let's update the file permissions and then try running the file. Use the following command to add execute permission to the file:

```
chmod +x generate_report.py
```

chmod +x generate_report.py

Now test the function by running the file using the following command:

```
./generate_report.py
```

The list employees_list within the script should return the list of dictionaries as shown below.

Click *Check my progress* to verify the objective.

Convert employee data to dictionary

Check my progress

Note: You can now remove the print statements once you get the desired output and have been assessed for this section.

Process employee data

The second function *process_data()* should now receive the list of dictionaries, i.e.,

Process employee data

The second function *process_data()* should now receive the list of dictionaries, i.e., employee_list as a parameter and return a dictionary of **department:amount**.

Open the file generate_report.py to define the function.

```
nano generate_report.py
```

```
def process_data(employee_list):
```

This function needs to pass the `employee_list`, received from the previous section, as a parameter to the function.

Now, initialize a new list called department_list, iterate over employee_list, and add only the departments into the **department_list**.

```
  department_list = []
  for employee_data in employee_list:
    department_list.append(employee_data['Department'])
```

The department_list should now have a redundant list of all the department names. We now have to remove the redundancy and return a dictionary. We will return this dicationary in the format **department:amount**, where amount is the number of employees in that particular department.

```
  department_data = {}
  for department_name in set(department_list):
    department_data[department_name] =
department_list.count(department_name)
  return department_data
```

in the format **department:amount**, where amount is the number of employees in that particular department.

This uses the set() method, which converts iterable elements to distinct elements.

Now, call this function by passing the `employee_list` from the previous section. Then, save the output in a variable called `dictionary`. Print the variable `dictionary`.

```
dictionary = process_data(employee_list)
print(dictionary)
```

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

Now test the function by running the file using the following command:

```
./generate_report.py
```

This should return a dictionary in the format **department: amount**, as shown below.

Click *Check my progress* to verify the objective.

Process employee data

Check my progress

Generate a report

Next, we will write the function *write_report*. This function writes a dictionary of **department: amount** to a file.

The report should have the format:

<department1>: <amount1>

<department2>: <amount2>

Lets open **generate_report.py** file to define the function.

```
nano generate_report.py
```

```
def write_report(dictionary, report_file):
```

This function requires a `dictionary`, from the previous section, and `report_file`, an output file to generate report, to both be passed as parameters.

You will use the `open()` function to open a file and return a corresponding file object. This function requires file path and file mode to be passed as parameters. The file mode is 'r' (reading) by default, so you should now explicitly pass 'w+' mode (open for reading and writing, overwriting a file) as a parameter.

Once you open the file for writing, iterate through the dictionary and use `write()` on the file to store the data.

```
with open(report_file, "w+") as f:
  for k in sorted(dictionary):
    f.write(str(k)+':'+str(dictionary[k])+'\n')
  f.close()
```

```
with open(report_file, "w+") as f:
  for k in sorted(dictionary):
    f.write(str(k)+':'+str(dictionary[k])+'\n')
  f.close()
```

Now call the function *write_report()* by passing a `dictionary` variable from the previous section and also passing a report_file. The report_file passed within this function should be similar to */home/<username>/data/report.txt*. Replace `<username>` with the one mentioned in Connection Details Panel at left-hand side.

```
write_report(dictionary, '<report_file>')
```

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

Let's execute the script now.

```
./generate_report.py
```

This script does not generate any output, but it creates a new file named **report.txt** within the **data** directory. This report.txt file should now have the count of people in each department.

Navigate to the `data` directory and list the files. You should see a new file named **report.txt**.

```
cd ~/data
```

```
ls
```

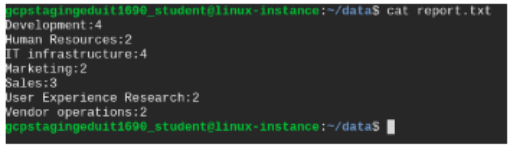
```
cd ~/data
```

```
ls
```

To view the generated report file, use the following command:

```
cat report.txt
```

The report file should be similar to the below image.



Click *Check my progress* to verify the objective.

Generate a report

Check my progress

Congratulations!

You successfully wrote a Python script that achieves two tasks. First, it reads a CSV file

Congratulations!

You successfully wrote a Python script that achieves two tasks. First, it reads a CSV file containing a list of the employees in the organization. Second, it generates a report of the number of people in each department in a plain text file.

Creating reports using Python is a very useful tool in IT support. You will likely complete similar tasks regularly throughout your career, so feel free to go through this lab more than once. Remember, practice makes perfect.

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab