

Editing Files using Substrings

 googlecoursera.qwiklabs.com/focuses/47747

Introduction

In this lab, you'll change the username of your coworker Jane Doe from "**jane**" to "**jd**" in compliance with company's naming policy. The username change has already been done. However, some files that were named with Jane's previous username "**jane**" haven't been updated yet. To help with this, you'll write a bash script and a Python script that will take care of the necessary rename operations.

What you'll do

- Practice using the cat, grep, and cut commands for file operations
- Use > and >> commands to redirect I/O stream
- Replace a substring using Python
- Run bash commands in Python

You'll have 90 minutes to complete this lab.

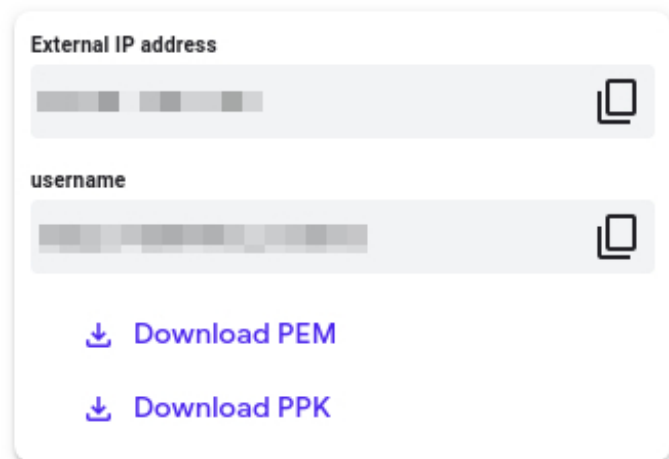
Start the lab

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).

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



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**.

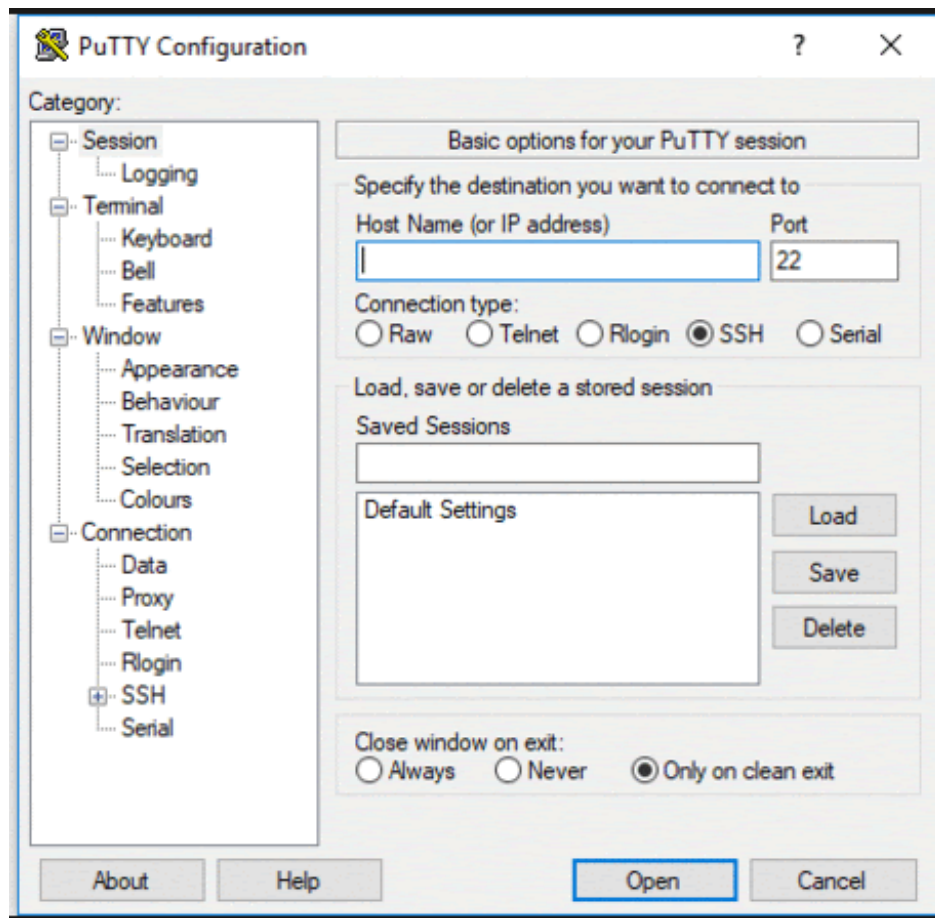
Connect to your VM using SSH and PuTTY

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

Download PEM

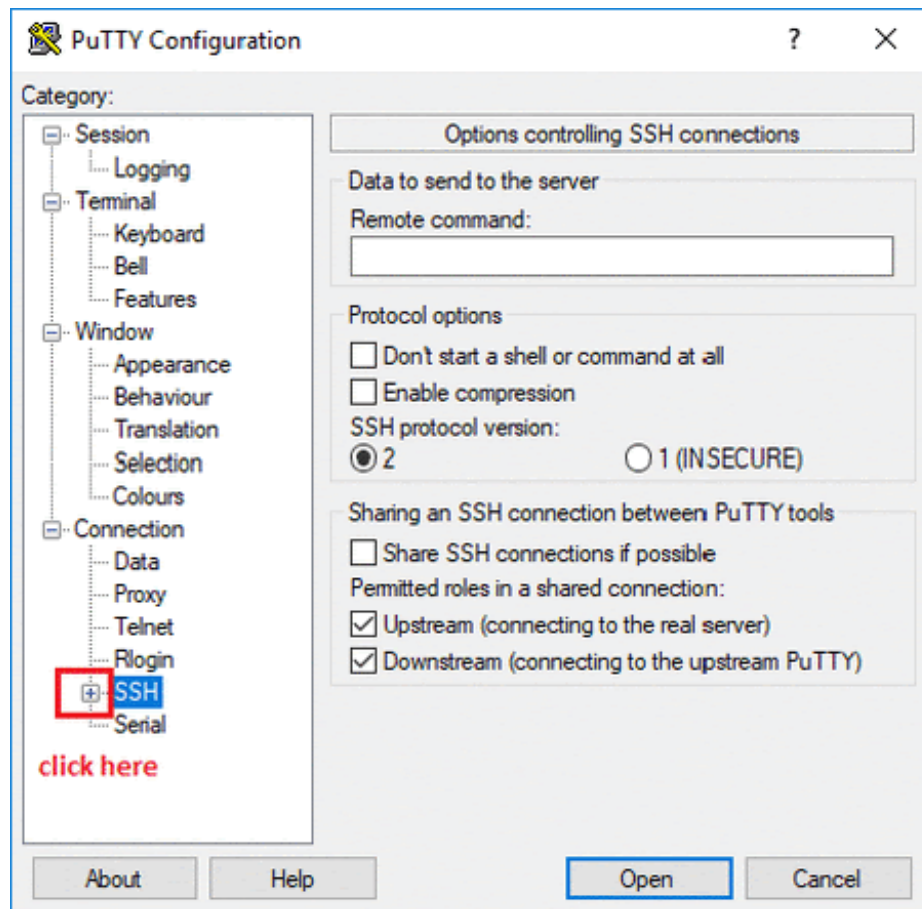
Download PPK

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-XXXXX.pem**

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



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



```
gcps$ 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:vrz8b4aYUtruFh0A6wZn60zy1oqqPEfh931o1vx1Tm8.
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]**.

A screenshot of the 'New Connection' dialog box in a Secure Shell application. The dialog has a title bar '[New Connection]'. Below the title bar is a text field 'username@hostname or free form text'. Underneath are three input fields: 'username', 'hostname', and 'port'. Below these is a text field 'SSH relay server options'. Further down is a section with 'Identity:' followed by a dropdown menu showing '[default]' and an 'Import...' button. Below that is 'SSH Arguments:' followed by a text field 'extra command line arguments'. Then 'Current profile:' followed by a dropdown menu showing 'default'. Finally, 'Mount Path:' followed by a text field 'the default path is the user's home directory'. At the bottom left are buttons '[DEL] Delete' and 'Options'. At the bottom right are buttons 'SFTP Mount', 'SFTP', '[ENTER] Connect'.

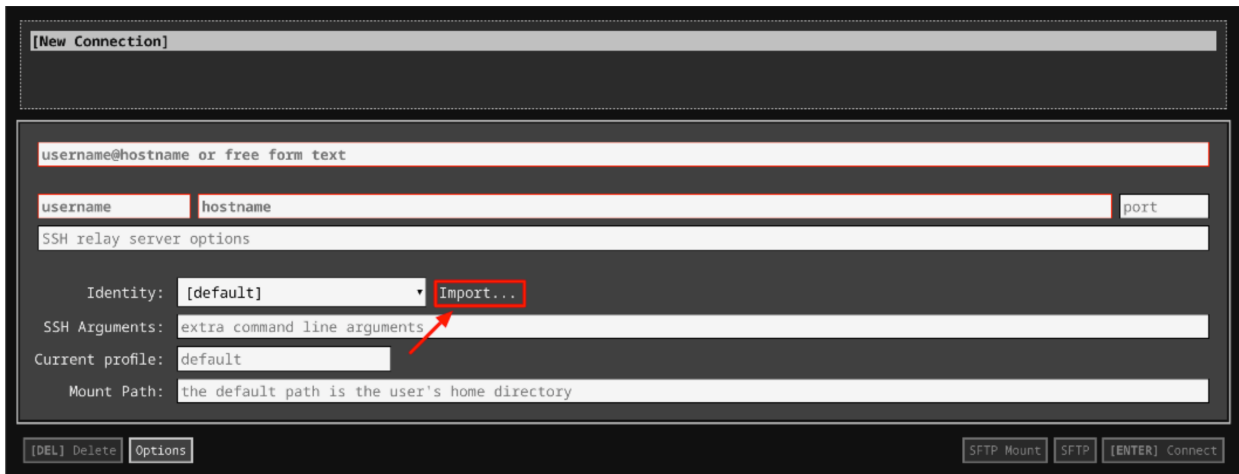
3. In the **username** section, enter the username given in the Connection Details Panel of the lab. And for the **hostname** section, enter the external IP of your VM instance that is mentioned in the Connection Details Panel of the lab.

A 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 bottom left towards the 'Import...' button.

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

For this lab, you should have a sound knowledge of these Linux commands:

- cat
- grep
- cut

cat:

The **cat** command allows us to create single or multiple files, view the contents of a file, concatenate files, and redirect output in terminal or other files.

Syntax:

cat [file]



grep:

The **grep** command, which stands for "global regular expression print", processes text line-by-line and prints any lines that match a specified **pattern**.

Syntax:

```
grep [pattern] [file-directory]
```



Here, **[file-directory]** is the path to the directory/folder where you want to perform a search operation. The **grep** command is also used to search text and match a string or pattern within a file.

Syntax:

```
grep [pattern] [file-location]
```



cut:

The **cut** command extracts a given number of characters or columns from a file. A delimiter is a character or set of characters that separate text strings.

Syntax:

```
cut [options] [file]
```



For delimiter separated fields, the **-d** option is used. The **-f** option specifies the field, a set of fields, or a range of fields to be extracted.

Syntax:

```
cut -d [delimiter] -f [field number]
```



Linux I/O Redirection

Redirection is defined as switching standard streams of data from either a user-specified source or user-specified destination. Here are the following streams used in I/O redirection:


- Redirection into a file using **>**
- Append using **>>**

Redirection into a file

Each stream uses redirection commands. A single greater than sign (**>**) or a double greater than sign (**>>**) can be used to redirect standard output. If the target file doesn't exist, a new file with the same name will be created.

Commands with a single greater than sign (>) **overwrite** existing file content.

```
cat > [file]
```



Commands with a double greater than sign (>>) **do not overwrite** the existing file content, but it will **append** to it.

```
cat >> [file]
```



So, rather than creating a file, the >> command is used to append a word or string to the existing file.

Exercise

The Scenario

Your coworker Jane Doe currently has the username "jane" but she needs to it to "jdoe" to comply with your company's naming policy. This username change has already been done. However, some files that were named with Jane's previous username "jane" haven't been updated. For example, "jane_profile_07272018.doc" needs to be updated to "jdoe_profile_07272018.doc".

Navigate to **data** directory by using the following command:

```
cd data
```



You can list the contents of the directory using the **ls** command. This directory contains a file named **list.txt**. You will also find some other files within this directory.

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

```
cat list.txt
```



Output:

```
student-00-d33bd0a2d20e@linux-instance:~/data$ cat list.txt
001 jane /data/jane_profile_07272018.doc
002 kwood /data/kwood_profile_04022017.doc
003 pchow /data/pchow_profile_05152019.doc
004 janez /data/janez_profile_11042019.doc
005 jane /data/jane_pic_07282018.jpg
006 kwood /data/kwood_pic_04032017.jpg
007 pchow /data/pchow_pic_05162019.jpg
008 jane /data/jane_contact_07292018.csv
009 kwood /data/kwood_contact_04042017.csv
010 pchow /data/pchow_contact_05172019.csv
```

This file contains three columns: line number, username, and full path to the file.

You could also view the complete `/data` directory using the `ls` command.

```
ls
```



Let's try out the commands we learned in the previous section to catch all the "jane" lines.

```
grep 'jane' ../data/list.txt
```



This returns all the files with the pattern "jane". It also matches the file that has string "janez" within it.

```
student-00-d33bd0a2d20e@linux-instance:~/scripts$ grep 'jane' ../data/list.txt
001 jane /data/jane_profile_07272018.doc
004 janez /data/janez_profile_11042019.doc
005 jane /data/jane_pic_07282018.jpg
008 jane /data/jane_contact_07292018.csv
```

Now, we'll list only the files containing the string "jane" and not include "janez".

```
grep ' jane ' ../data/list.txt
```



This now returns only files containing the string "jane".

```
student-00-d33bd0a2d20e@linux-instance:~/scripts$ grep ' jane ' ../data/list.txt
001 jane /data/jane_profile_07272018.doc
005 jane /data/jane_pic_07282018.jpg
008 jane /data/jane_contact_07292018.csv
```

Next, we'll use the **cut** command with **grep** command. For cut command, we'll use the whitespace character (' ') as a delimiter (denoted by `-d`) since the text strings are separated by spaces within the **list.txt** file. We'll also fetch results by specifying the fields using `-f` option.

Let's fetch the different fields (columns) using `-f` flag :

```
grep "jane" ../data/list.txt | cut -d ' ' -f 1
```



Output:

```
student-00-d33bd0a2d20e@linux-instance:~/scripts$ grep "jane" ../data/list.txt | cut -d ' ' -f1
001
005
008
```

```
grep "jane" ../data/list.txt | cut -d ' ' -f 2
```



Output:

```
student-00-d33bd0a2d20e@linux-instance:~/scripts$ grep "jane" ../data/list.txt | cut -d ' ' -f2
jane
jane
jane
```

```
grep "jane" ../data/list.txt | cut -d ' ' -f 3
```



Output:

```
student-00-d33bd0a2d20e@linux-instance:~/scripts$ grep "jane" ../data/list.txt | cut -d ' ' -f3
/data/jane_profile_07272018.doc
/data/jane_pic_07282018.jpg
/data/jane_contact_07292018.csv
```

You can also return a range of fields together by using:

```
grep "jane" ../data/list.txt | cut -d ' ' -f 1-3
```



To return a set of fields together:

```
grep "jane" ../data/list.txt | cut -d ' ' -f 1,3
```



Test command

We'll now use the **test** command to test for the presence of a file. The command **test** is a command-line utility on Unix-like operating systems that evaluates conditional expressions.

The syntax for this command is:

```
test EXPRESSION
```



We'll use this command to check if a particular file is present in the file system. We do this by using the **-e** flag. This flag takes a filename as a parameter and returns True if the file exists.

We'll check the existence of a file named **jane_profile_07272018.doc** using the following command:

```
if test -e ~/data/jane_profile_07272018.doc; then echo "File exists"; else echo "File doesn't exist"; fi
```



Output:

```
student-00-b5f5a7af792f@linux-instance:~/scripts$ if test -e ~/data/jane_profile_07272018.doc; then echo "File exists"; else echo "File doesn't exist"; fi
File exists
```

Create a file using a Redirection operator

We'll now use the redirection operator (**>**) to create an empty file simply by specifying the file name. The syntax for this is:

```
> [file-name]
```



Let's create a file named **test.txt** using the redirection operator.

```
> test.txt
```



Output:

```
student-00-b5f5a7af792f@linux-instance:~$ > test.txt
student-00-b5f5a7af792f@linux-instance:~$ ls
data  scripts  test.txt
```

To append any string to the **test.txt** file, you can use another redirection operator (**>>**).

```
echo "I am appending text to this test file" >> test.txt
```



You can view the contents of the file at any time by using the **cat** command.

```
cat test.txt
```



Output:

```
student-00-b5f5a7af792f@linux-instance:~$ echo "I am appending text into the test file" >> test.txt
student-00-b5f5a7af792f@linux-instance:~$ cat test.txt
I am appending text into the test file
```

Iteration

Another important aspect of a scripting language is iteration. Iteration, in simple terms, is the repetition of a specific set of instructions. It's when a set of instructions is repeated a number of times or until a condition is met. And for this process, bash script allows three different iterative statements:

- **For:** A for loop repeats the execution of a group of statements over a set of items.
- **While:** A while loop executes a set of instructions as long as the control condition remains true.
- **Until:** An until loop executes a set of instructions as long as the control condition remains false.

Let's now iterate over a set of items and print those items.

```
for i in 1 2 3; do echo $i; done
```



Output:

```
student-00-b5f5a7af792f@linux-instance:~$ for i in 1 2 3; do echo $i; done
1
2
3
```

Find files using bash script

In this section, you are going to write a script named **findJane.sh** within the `scripts` directory.

This script should catch all "jane" lines and store them in another text file called **oldFiles.txt**. You will complete the script using the command we practiced in earlier sections. Don't worry, we'll guide you throughout the whole process.

Navigate to `/scripts` directory and create a new file named **findJane.sh**.

```
cd ~/scripts
```



```
nano findJane.sh
```



Now, add the shebang line.

```
#!/bin/bash
```

Create the text file **oldFiles.txt** and make sure it's empty. This **oldFiles.txt** file should save files with username "**jane**".

```
> oldFiles.txt
```

Now, search for all lines that contain the name " **jane** " and save the file names into a variable. Let's call this variable **files**, we will refer to it with that name later in the lab.

Since none of the files present in the file **list.txt** are available in the file system, check if file names present in **files** variable are actually present in the file system. To do this, we'll use the **test** command that we practiced in the previous section.

Now, iterate over the **files** variable and add a test expression within the loop. If the item within the files variable passes the test, add/append it to the file **oldFiles.txt**.

Once you have completed writing the bash script, save the file by clicking Ctrl-o, Enter key, and Ctrl-x.

Make the file executable using the following command:

```
chmod +x findJane.sh
```

Run the bash script **findJane.sh**.

```
./findJane.sh
```

This will generate a new file named **oldFiles.txt**, which consists of all the files containing the name "jane".

Use the **cat** command followed by the file name to view the contents of the newly generated file.

```
cat oldFiles.txt
```

Output:

```
student-01-837a3caffc82@linux-instance:~/scripts$ cat oldFiles.txt
/home/student-01-837a3caffc82/data/jane_profile_07272018.doc
/home/student-01-837a3caffc82/data/jane_contact_07292018.csv
```

Click *Check my progress* to verify the objective. Find files using bash script

Rename files using Python script

In this section, you are going to write a Python script, **changeJane.py**, that takes **oldFiles.txt** as a command line argument and then renames files with the new username "jdoe". You will be completing the script, but we will guide throughout the section.

Create a Python script **changeJane.py** under **/scripts** directory using nano editor.

```
nano changeJane.py
```



Add the shebang line.

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



Now, import the necessary Python module to use in the Python script.

```
import sys
import subprocess
```



The sys (System-specific parameters and functions) module provides access to some variables used or maintained by the interpreter and to functions that interact with the interpreter. The subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and get their return codes.

Continue writing the script to achieve the goal!

Since **oldFiles.txt** is passed as a command line argument, it's stored in the variable `sys.argv[1]`. Open the file from the first argument to read its contents using `open()` method. You can either assign it to a variable or use a **with** block. Hint: traverse each line in the file using `readlines()` method. Use `line.strip()` to remove any whitespaces or newlines and fetch the old name.

Once you have the old name, use `replace()` function to replace "jane" with "jdoe". This method replaces occurrences of any older substring with the new substring. The old and new substrings are passed as parameters to the function. Therefore, it returns a string where all occurrences of the old substring is replaced with the new substring.

Syntax:

```
string.replace(old_substring, new_substring)
```



Now, invoke a subprocess by calling **run()** function. This function takes arguments used to launch the process. These arguments may be a list or a string.

In this case, you should pass a list consisting of the command to be executed, followed by arguments to the command.

Use the **mv** command to rename the files in the file system. This command moves a file or directory. It takes in source file/directory and destination file/directory as parameters. We'll move the file with old name to the same directory but with a new name.

Syntax:

```
mv source destination
```



Now it must be clear. You should pass a list consisting of the **mv** command, followed by the variable storing the old name and new name respectively to the **run()** function within the subprocess module.

Close the file that was opened at the beginning.

```
f.close()
```



Make the file executable using the following command:

```
chmod +x changeJane.py
```



Run the script and pass the file **oldFiles.txt** as a command line argument.

```
./changeJane.py oldFiles.txt
```



Navigate to the **/data** directory and use the **ls** command to view renamed files.

```
cd ~/data
```



```
ls
```




```
student-00-b5f5a7af792f@linux-instance:~/data$ ls
janez_profile_11042019.doc  jdoe_profile_07272018.doc  kwood_profile_04022017.doc  pchow_pic_05162019.jpg
jdoe_contact_07292018.csv  kwood_pic_04032017.jpg    list.txt
```

Click *Check my progress* to verify the objective. Rename files using Python script

Congratulations!

Congrats! You've successfully renamed files containing named "jane" with "jdoe" and we bet you're feeling much more comfortable performing operations using cat, grep, cut, and I/O stream commands. On top of that, you now know how to use conditional statements and iteration through bash scripts. Working with bash and Python scripting simultaneously will be super helpful throughout your IT specialist career. Great job completing this lab!

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.