Editing Files using Substrings

Introduction

In this lab, you'll change the username of your coworker Jane Doe from "**jane**" to "**jdoe**" 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

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:

```
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.

```
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".

```
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:

```
001
005
008
```

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

Output:

```
jane
jane
jane
```

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

Output:

```
/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:

```
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:

```
jane_contact_07292018.csv    jane_profile_07272018.doc    janez_profile_11042019.doc
kwood_pic_04032017.jpg  kwood_profile_04022017.doc  list.txt  pchow_pic_05162019.jpg
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:

```
I am appending text to this 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:

```
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.

```
bash
#!/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:

```
/home/student-02-196e48437fa8/data/jane_profile_07272018.doc
/home/student-02-196e48437fa8/data/jane_contact_07292018.csv
```

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
```

```
janez_profile_11042019.doc    jdoe_contact_07292018.csv    jdoe_profile_07272018.doc
kwood_pic_04032017.jpg  kwood_profile_04022017.doc  list.txt  pchow_pic_05162019.jpg
test.txt
```