CSCE 3600: Systems Programming

Recitation Assignment 3 – Using sed and gawk

Due: 11:59 PM on Friday, September 24, 2021

RECITATION DESCRIPTION:

The purpose of this laboratory assignment is to get you more familiar with two regex-based utilities, **sed** and g**awk**. In this recitation assignment, you will perform several operations using these utilities to manipulate files and create reports (i.e., new files) from these changes. You will turn in a typescript of your terminal session that can be done using the **script** command. The syntax of the script command is as follows:

```
   script [options] filename
```

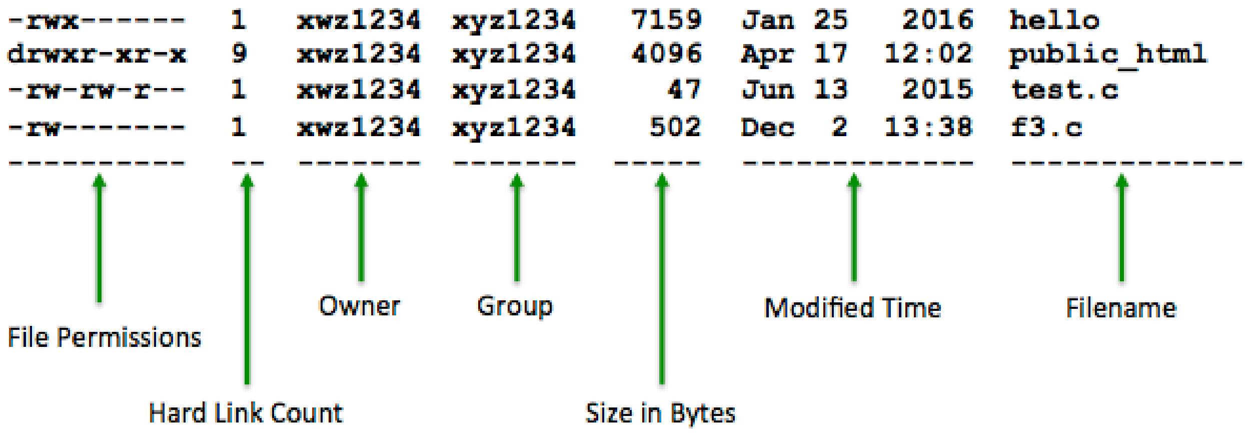
Before you begin your work, you will issue the **script** command with your desired filename (using no options). You will then complete the tasks for **sed** and **awk** given in this recitation assignment. Then, once you are finished, you will issue the **exit** command, which will actually just exit from the script (not your login session) with your typescript of your terminal session. *Note that you will submit this typescript file as part of this recitation, so you must issue the script command before completing the following.*

The sed Stream Editor

First, find one of your directories that contains plenty of files (at least 20) and then in that directory create a list of files by issuing the following command from your command prompt as follows:

```
   $ ls -l > sedfile.txt
```

This will create a list of your files using the long listing format as shown here:



```
-rwx------   1  xwz1234  xyz1234   7159  Jan 25   2016  hello
drwxr-xr-x   9  xwz1234  xyz1234   4096  Apr 17  12:02  public_html
-rw-rw-r--   1  xwz1234  xyz1234     47  Jun 13   2015  test.c
-rw-------   1  xwz1234  xyz1234    502  Dec  2  13:38  f3.c
----------  --  -------  -------  -----  -------------  -------------
```

Be sure that you have plenty of files (at least 20) or various types and age in this file listing so that you have something to work with (i.e., an empty or sparse directory would result in a less than acceptable solution).

The first thing we want to do is change the file type of ordinary files in your file listing (as shown through the first character in the file permissions) from “–” to “R”. We can do this very easily using regex with the following command:

```
$ sed –i "s/^-/R/g" sedfile.txt
```

This does an in-place substitution to your **sedfile.txt** file for all lines that start with "–" directed by the regex metacharacter "^".

Now rather than printing out the entire file (which could be very long) to show the resulting changes, let’s just print out the first 10 lines of the file using **sed**. We do this with the following command:

```
$ sed –n 1,10p sedfile.txt
```

The "–n" is needed to only print out the first 10 lines as the default is to print out every line.

You may have noticed that the first line of your **sedfile.txt** file starts with the word "total" followed by a number and does not follow the listing format shown in the above figure. To just show the actual first 10 files of this listing, we can easily modify the previous command by using "2,11p" instead. But suppose we just want to delete this first line instead? We can do this with the following command:

```
$ sed –i 1d sedfile.txt
```

This line simply deletes in-place the first line of your **sedfile.txt** file.

Now, we want to be able to add a new line of text to your file after a pattern match is found. Look at the fifth line of your file and find a unique pattern specific to this line only. It may be something like the date, size in bytes, or the file name. To add this new line after this fifth line in your file, issue the following command:

```
$ sed –i '/YOUR PATTERN/a This is a new line' sedfile.txt
```

Instead of "YOUR PATTERN", you would enter the unique pattern specific to the fifth line of your file. Print out the first 10 lines of your **sedfile.txt** file again to verify that these changes (deleting the first line and adding a new line) worked:

```
$ sed –n 1,10p sedfile.txt
```

We’re finished with **sed** now. **sed** is a powerful stream editor. There are many more things that you can do with **sed**, but for now let’s change our focus to **gawk**! *Do NOT type exit yet as we will continue and include the work on* ***gawk*** *as well in this file.*

The gawk Programming Language

This time, we’re going to focus on process instead of files. Now, create a list of running processes on your system by issuing the following command from your command prompt as follows:

```
$ ps –ef > gawkfile.txt
```

The first thing we want to do is check out that this command worked by printing the first few lines of the file. We can do this by using the NR system variable as follows:

```
$ gawk 'NR <= 10' gawkfile.txt
```

This should display the first 10 lines of your g**awkfile.txt** file, showing a few of the processes running on your system. You can find out more about the fields shown in the command by issuing the **info ps** command (and moving to about 80% down on the document), which may be helpful in understanding some of the manipulations and reports that we will make. We will cover processes in detail a little later on in this course.

So how many processes are running on your system? We can print out the number of records (i.e., lines) again making use of the NR system variable with the command:

```
$ gawk 'END {print NR}' gawkfile.txt
```

Since our g**awkfile.txt** file contains a header with the fields, we should have 1 less process running on our system.

Now, lets print out only those processes that you are running. This means that we want to print out some fields where the UID, the effective user ID of the process' owner, is your user ID. If you do not know your user ID, you can issue the **whoami** command at your command prompt to find out. Then, issue the following command:

```
$ gawk '/YOUR USER ID/{print NR, $0}' gawkfile.txt
```

Instead of "YOUR USER ID", you would enter your user ID that you found through the **whoami** command. Since we just matched on any field with your user ID, chances are you got one or more results where you are not the owner (unless you are running as root), so we want to make sure that we only find those processes where the UID (or field $1) is your user ID. So we rewrite the above command like this:

```
$ gawk '$1 ~ /YOUR USER ID/{print NR, $0}' gawkfile.txt
```

This ensures that only the processes that you own are printed!

We see that the first line of **gawkfile.txt** contains a header with most likely 8 fields (i.e., UID, PID, PPID, C, STIME, TTY, TIME, and CMD, though there might be some variation in different Linux distributions). However, because we use the default field separator of whitespace, some records might contain more fields. So, let's print out those records that have more than 8 fields! If we print out the record number (NR) and number of fields (NF) for those with more than 8 fields, we might see that some records have 9, 10, 11, or even more fields! Check it out with the following command:

```
$ gawk 'NF > 8 {print NR, NF}' gawkfile.txt
```

So how can we print all of the fields when it is variable and we don't know exactly how many fields each record has? Remembering that **gawk** is a programming language, we can add a simple **for** loop as in the following:

```
   $ gawk 'NF > 8 {for(i = 9; i <= NF; i++) print NR, NF,
$i}' gawkfile.txt
```

Notice, however, that each extra field is printed on a separate line.

Now, we want to do one last thing – we want to swap the PID (or field $2) and PPID (or field $3) columns so that each record lists the parents' process ID first. We can do this simply with the following:

```
   $ gawk 'BEGIN {OFS = "\t"}; {print $1, $3, $2, $4,    $5,
$6, $7, $8}' gawkfile.txt
```

A couple notes: (1) we added the OFS (Output Field Separator) to be a tab to help formatting a bit and (2) we only included the fixed number of 8 fields for simplicity. We can write these changes to a new file using redirection or in some newer versions of **gawk** we can use the "–i inplace" option to write these changes in-place to the same file.

At this point, if you typed the **script** command at the start of this part of the recitation, you should type **exit** now to save the typescript of your terminal session. This typescript file should be submitted for this lab.

REQUIREMENTS:

- Your typescript file will be graded based largely on whether you completed the work correctly on the CSE machines (e.g., `cse01`, `cse02`, …, `cse06`), so you should make sure you obtained meaningful results from each command.
- Although this assignment is to be submitted individually (i.e., each student will submit his/her own source code), you may receive assistance from your TA, IA (i.e., Grader), and even other classmates. Please remember that you are ultimately responsible for learning and comprehending this material as the recitation assignments are given in preparation for the minor assignments, which must be completed individually.

SUBMISSION:

- You will electronically submit your typescript file to the **Recitation 3** dropbox in Canvas by the due date and time. No late recitation assignments will be accepted.