CDOT
SICT AR Meeting Area
People

get involved with CDOT

as a Student
as an Open Source Community Member
as a Company

courses

BTC640
BTH740
BTP300
DPI908
DPS901
DPS905
DPS909
DPS911
DPS914
DPS915
DPS924
DPS931
EAC234
ECL500
GAM531
GAM666
GAM670
GPU610
LUX Program
MAP524
OOP344
OPS235
OPS245
OPS335
OPS345
OPS435
OPS445
OPS535
OPS635
OSD600
OSD700
OSL640
OSL740
OSL840
Real World Mozilla
RHT524
SBR600

Tutorial10: Sed & Awk Utilities

Contents [hide]

USING SED & AWK UTILTIES

Main Objectives of this Practice Tutorial

- Use the **sed** command to **manipulate text** contained in a file.
- List and explain several **addresses** and **instructions** associated with the **sed** command.
- Use the **sed** command as a **filter** with Linux pipeline commands.
- Use the **awk** command to **manipulate text** contained in a file.
- List and explain **comparison operators**, **variables** and **actions** associated with the **awk** command.
- Use the **awk** command as a **filter** with Linux pipeline commands.

Tutorial Reference Material

Course Notes	Linux Command/Shortcut Reference		YouTube Videos
Slides: • Week 10 Lecture 1 Notes: PDF | PPTX • Week 10 Lecture 2 Notes: PDF | PPTX	**Text Manipulation:** • Purpose of using the sed utility • Purpose of using the awk utility	**Commands:** • sed • awk	**Brauer Instructional Videos:** • Using the sed Utility • Using the awk Utility

SEC520
SPO600
SRT210
ULI101

course projects

Course Project List
Potential Course Projects
Contrib Opportunities

links

CDOT
Planet CDOT
FSOSS

Tools

What links here
Related changes
Special pages
Printable version
Permanent link
Page information

KEY CONCEPTS

Using the sed Utility

Usage:

```
Syntax: sed [-n] 'address instruction' filename
```

How it Works:

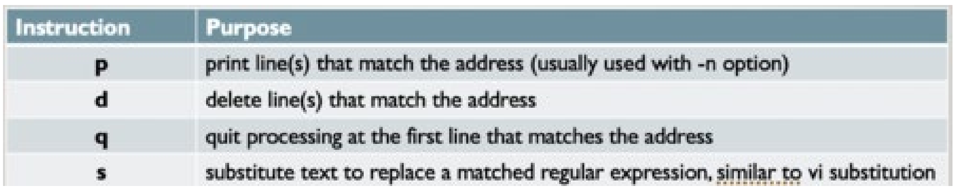
- The sed command reads all lines in the input file and will be exposed to the expression (i.e. area contained within quotes) one line at a time.
- The expression can be within single quotes or double quotes.
- The expression contains an address (match condition) and an instruction (operation).
- If the line matches the address, then it will perform the instruction.
- Lines will display be default unless the **–n** option is used to suppress default display

Address:

- Can use a line number, to select a specific line (for example: **5**)
- Can specify a range of line numbers (for example: **5,7**)
- Regular expressions are contained within forward slashes (e.g. /regular-expression/)
- Can specify a regular expression to select all lines that match a pattern (e.g **/^[0-9].*[0-9]$/**)
- If NO address is present, the instruction will apply to ALL lines

Instruction:

- **Action** to take for matched line(s)
- Refer to table on right-side for list of some **common instructions** and their purpose



Instruction	Purpose
p	print line(s) that match the address (usually used with -n option)
d	delete line(s) that match the address
q	quit processing at the first line that matches the address
s	substitute text to replace a matched regular expression, similar to vi substitution

Using the awk Utility

Usage:

```
awk [-F] 'selection-criteria {action}’ file-name
```

How It Works:

- The **awk** command reads all lines in the input file and will be exposed to the expression (contained within quotes) for processing.
- The **expression** (contained in quotes) represents **selection criteria**, and **action** to execute contained within braces **{}**
- if selection criteria is matched, then action (between braces) is executed.
- The **–F** option can be used to specify the default **field delimiter** (separator) character eg. **awk –F”;”** (would indicate a semi-colon delimited input file).

Selection Criteria

- You can use a regular expression, enclosed within slashes, as a pattern. For example: **/pattern/**
- The ~ operator tests whether a field or variable matches a regular expression. For example: **$1 ~ /^[0-9]/**
- The **!~** operator tests for no match. For example: **$2 !~ /line/**
- You can perform both numeric and string comparisons using relational operators (**>** , **>=** , **<** , **<=** , **==** , **!=**).
- You can combine any of the patterns using the Boolean operators **||** (OR) and **&&** (AND).
- You can use built-in variables (like NR or "record number" representing line number) with comparison operators.
  For example: **NR >=1 && NR <= 5**

Action (execution):

- Action to be executed is contained within braces **{}**
- The **print** command can be used to display text (fields).
- You can use parameters which represent fields within records (lines) within the expression of the awk utility.
- The parameter **$0** represents all of the fields contained in the record (line).
- The parameters **$1**, **$2**, **$3** … **$9** represent the first, second and third to the 9th fields contained within the record.
- Parameters greater than nine requires the value of the parameter to be placed within braces (for example: **${10}**,**${11}**,**${12}**, etc.)
- You can use built-in **variables** (such as **NR** or "record number" representing line number) eg. **{print NR,$0}** (will print record number, then entire record).

INVESTIGATION 1: USING THE SED UTILITY

ATTENTION: This online tutorial will be required to be completed by **Friday in week 11 by midnight** to obtain a grade of **2%** towards this course

In this investigation, you will learn how to manipulate text using the **sed** utility.

Perform the Following Steps:

1. **Login** to your matrix account and confirm you are located in your **home** directory.

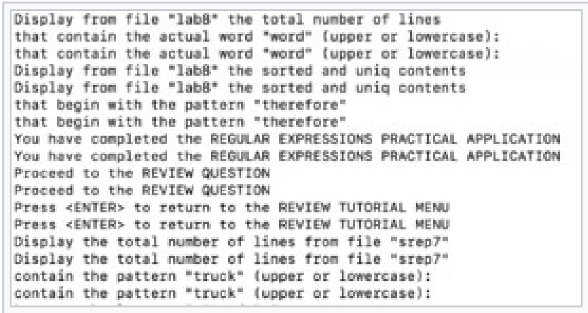
2. Issue a Linux command to create a directory called **sed**

3. Issue a Linux command to change to the **sed** directory and confirm that you are located in the **sed** directory.

4. Issue the following Linux command to download the data.txt file (**copy and paste** to save time):
   ```
   wget https://github.com/ULI101/labs/raw/main/data.txt
   ```

5. Issue the **more** command to quickly view the contents of the **data.txt** file.
   When finished, exit the more command by pressing the letter `q`

The **p** instruction with the **sed** command is used to **print** (i.e. *display*) the contents of a text file.



```
Display from file "lab8" the total number of lines
that contain the actual word "word" (upper or lowercase):
that contain the actual word "word" (upper or lowercase):
Display from file "lab8" the sorted and uniq contents
Display from file "lab8" the sorted and uniq contents
that begin with the pattern "therefore"
that begin with the pattern "therefore"
You have completed the REGULAR EXPRESSIONS PRACTICAL APPLICATION
You have completed the REGULAR EXPRESSIONS PRACTICAL APPLICATION
Proceed to the REVIEW QUESTION
Proceed to the REVIEW QUESTION
Press <ENTER> to return to the REVIEW TUTORIAL MENU
Press <ENTER> to return to the REVIEW TUTORIAL MENU
Display the total number of lines from file "srep7"
Display the total number of lines from file "srep7"
contain the pattern "truck" (upper or lowercase):
contain the pattern "truck" (upper or lowercase):
```

Issuing the **p** instruction without using the **-n** option (to suppress original output) will display lines twice.

6. Issue the following Linux command:

```
sed 'p' data.txt
```

NOTE: You should notice that each line appears twice.

You have completed the COMPLEX REGULAR EXPRESSIONS REVIEW
You have completed the COMPLEX REGULAR EXPRESSIONS REVIEW
You can Send Feedback and exit the tutorial
You can Send Feedback and exit the tutorial
Press <ENTER> to return to the REVIEW TUTORIAL MENU
Press <ENTER> to return to the REVIEW TUTORIAL MENU
"A line totally contained within double quotes"
"A line totally contained within double quotes"

The reason why standard output appears twice is that the sed command (without the **-n option**) displays all lines regardless of an address used.

We will use **pipeline commands** to both display stdout to the screen and save to files for confirmation of running these pipeline commands when run a **checking-script** later in this investigation.

7. Issue the following Linux pipeline command:

```
sed -n 'p' data.txt | tee sed-1.txt
```

Display all lines from ALL files in the current
You have completed the COMPLEX REGULAR EXPRESSIONS REVIEW
You can Send Feedback and exit the tutorial
Press <ENTER> to return to the REVIEW TUTORIAL MENU
"A line totally contained within double quotes"

What do you notice? You should see only one line.

You can specify an **address** to display lines using the sed utility (eg. *line #*, **line #s** or range of **line #s**).

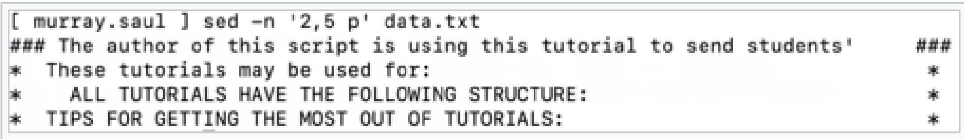
8. Issue the following Linux pipeline command:

```
sed -n '1 p' data.txt | tee sed-2.txt
```

Introductory Linux (Unix) course OPS224. The structure of the

You should see the first line of the text file displayed.
What other command is used to only display the first line in a file?

9. Issue the following Linux pipeline command:



```
[ murray.saul ] sed -n '2,5 p' data.txt
### The author of this script is using this tutorial to send students'     ###
*  These tutorials may be used for:                                         *
*    ALL TUTORIALS HAVE THE FOLLOWING STRUCTURE:                            *
*  TIPS FOR GETTING THE MOST OUT OF TUTORIALS:                              *
```

Using the sed command to display a **range** of lines.

```
sed -n '2,5 p' data.txt | tee sed-3.txt
```

The author of this script is using this tutorial to send students'
* These tutorials may be used for: *
* ALL TUTORIALS HAVE THE FOLLOWING STRUCTURE: *
* TIPS FOR GETTING THE MOST OUT OF TUTORIALS: *

What is displayed? How would you modify the sed command to display the line range 10 to 50?

sed -n '10,50 p' data.txt

The **s** instruction is used to **substitute** text
(a similar to method was demonstrated in the vi editor in tutorial 9).

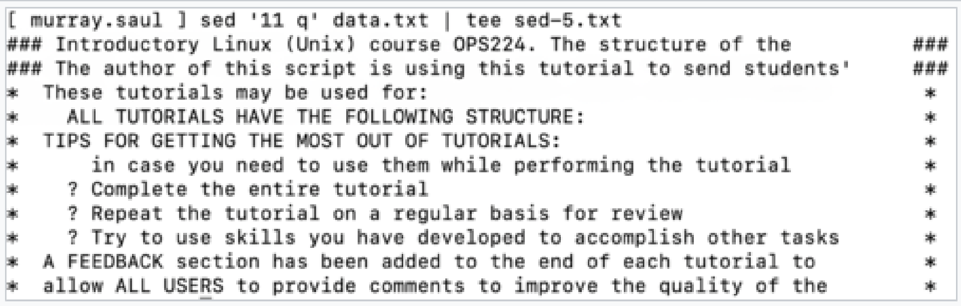
10. Issue the following Linux pipeline command:

```
sed '2,5 s/TUTORIAL/LESSON/g' data.txt | tee sed-4.txt | more
```

All TUTORIAL are substituted by LESSON

What do you notice? View the original contents of lines 2 to 5 in the **data.txt** file in another shell to confirm that the substitution occurred.

The **q**



```
[ murray.saul ] sed '11 q' data.txt | tee sed-5.txt
### Introductory Linux (Unix) course OPS224. The structure of the          ###
### The author of this script is using this tutorial to send students'     ###
*  These tutorials may be used for:                                          *
*    ALL TUTORIALS HAVE THE FOLLOWING STRUCTURE:                             *
*  TIPS FOR GETTING THE MOST OUT OF TUTORIALS:                               *
*      in case you need to use them while performing the tutorial            *
*    ? Complete the entire tutorial                                          *
*    ? Repeat the tutorial on a regular basis for review                     *
*    ? Try to use skills you have developed to accomplish other tasks        *
*  A FEEDBACK section has been added to the end of each tutorial to          *
*  allow ALL USERS to provide comments to improve the quality of the         *
```

Using the sed command with the **-q** option to display up to a line number, then quit.

instruction terminates or **quits** the execution of the sed utility as soon as it is read in a particular line or matching pattern.

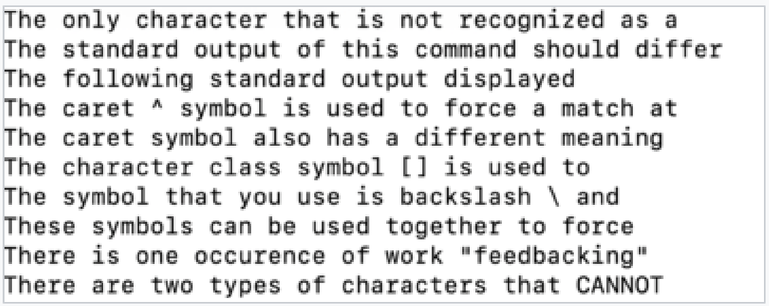
11. Issue the following Linux pipeline command:
    `sed '11 q' data.txt | tee sed-5.txt`

    What did you notice? How many lines were displayed before the sed command exited? First 11 lines in total are displayed

    You can use **regular expressions** to select lines that match a pattern. In fact, the sed command was one of the first Linux commands that used regular expression.

    The rules remain the same for using regular expressions as demonstrated in **tutorial 9** except the regular expression must be contained within **forward slashes** (eg. `/regexp/`).

12. Issue the following Linux pipeline command:
    `sed -n '/^The/ p' data.txt | tee sed-6.txt`



```
The only character that is not recognized as a
The standard output of this command should differ
The following standard output displayed
The caret ^ symbol is used to force a match at
The caret symbol also has a different meaning
The character class symbol [] is used to
The symbol that you use is backslash \ and
These symbols can be used together to force
There is one occurence of work "feedbacking"
There are two types of characters that CANNOT
```

Using the sed command using regular expressions with **anchors**.

    What do you notice?
Display the lines begin with **"The"**

13. Issue the following Linux pipeline command:
    `sed -n '/d$/ p' data.txt | tee sed-7.txt`

    What do you notice? Display lines that end with **"d"**

    The **sed** utility can also be used as a **filter** to manipulate text that was generated from Linux commands.

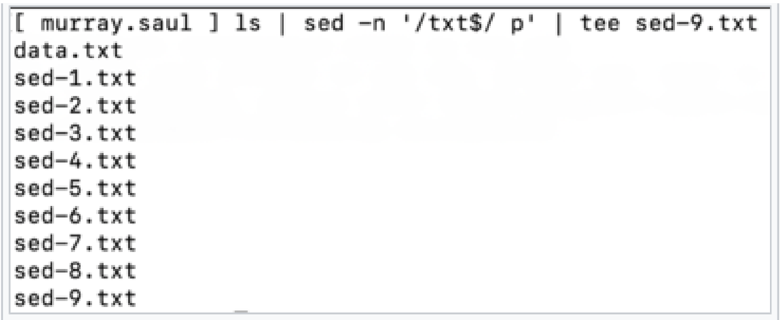
14. Issue the following Linux pipeline

command:

```
who | sed -n
'/^[a-m]/ p' |
tee sed-8.txt |
more
```

```
kchan151 pts/0        2023-03-29 16:30 (10.29.0.181)
cdrosero pts/2        2023-03-29 16:22 (10.29.3.203)
cprajapati2 pts/4     2023-03-29 16:27 (10.248.186.12)
gkaur532 pts/5        2023-03-29 16:36 (10.248.62.29)
iali74   pts/7        2023-03-29 15:06 (10.248.29.217)
fjahanian pts/9       2023-03-29 15:52 (10.29.0.173)
dyu55    pts/13       2023-03-29 14:12 (10.29.0.233)
```

What did you notice?



```
[ murray.saul ] ls | sed -n '/txt$/ p' | tee sed-9.txt
data.txt
sed-1.txt
sed-2.txt
sed-3.txt
sed-4.txt
sed-5.txt
sed-6.txt
sed-7.txt
sed-8.txt
sed-9.txt
```

Using the sed command with **pipeline** commands.

15. Issue the following Linux pipeline command:
    `ls | sed -n '/txt$/ p' | tee sed-9.txt`

```
data.txt
sed-1.txt
sed-2.txt
sed-3.txt
sed-4.txt
sed-5.txt
sed-6.txt
sed-7.txt
sed-8.txt
sed-9.txt
```

    What did you notice?

16. Issue the following to run a checking script:
    `~uli101/week10-check-1`

    If you encounter errors, make corrections and **re-run** the checking script until you receive a congratulations message, then you can proceed.

In the next investigation, you will learn how to manipulate text using the **awk** utility.

INVESTIGATION 2: USING THE AWK UTILITY

In this investigation, you will learn how to use the awk utility to manipulate text and generate reports.

Perform the Following Steps:

1. Change to your **home** directory and issue a command to **confirm** you are located in your *home* directory.

2. Issue a Linux command to create a directory called **awk**

3. Issue a Linux command to change to the **awk** directory and confirm you are located in the **awk** directory.

    Let's download a database file that contains information regarding classic cars.

4. Issue the following linux command (**copy and paste** to save time):
    `wget https://github.com/ULI101/labs/raw/main/cars.txt`

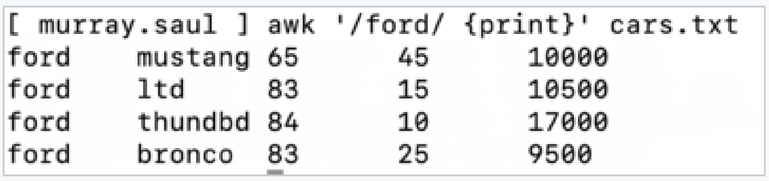
5. Issue the **cat** command to quickly view the contents of the **cars.txt** file.

```
plym    fury     77   73   2500
chevy   nova     79   60   3000
ford    mustang  65   45   10000
volvo   gl       78   102  9850
ford    ltd      83   15   10500
chevy   nova     80   50   3500
fiat    600      65   115  450
honda   accord   81   30   6000
ford    thundbd  84   10   17000
toyota  tercel   82   180  750
chevy   impala   65   85   1550
ford    bronco   83   25   9500
```

The **"print"** action (command) is the default action of awk to print all selected lines that match a **pattern**.

This **action** (contained in braces) can provide more options such as printing **specific fields** of selected lines (or records) from a database.

6. Issue the following linux command all to display all lines (i.e. records) in the **cars.txt** database that matches the pattern (or "make") called **ford**:
   ```
   awk '/ford/ {print}' cars.txt
   ```



```
[ murray.saul ] awk '/ford/ {print}' cars.txt
ford    mustang 65      45      10000
ford    ltd     83      15      10500
ford    thundbd 84      10      17000
ford    bronco  83      25      9500
```

Using the awk command to display matches of the pattern **ford**.

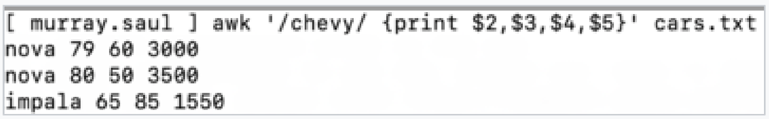
   We will use **pipeline commands** to both display stdout to the screen and save to files for confirmation of running these pipeline commands when run a **checking-script** later in this investigation.

7. Issue the following linux pipeline command all to display records in the **cars.txt** database that contain the pattern (i.e. make) **ford**:
   ```
   awk '/ford/' cars.txt | tee awk-1.txt
   ```

   What do you notice? You should notice ALL lines displayed without using **search criteria**.

   You can use *builtin* **variables** with the **print** command for further processing.
   We will discuss the following variables in this tutorial:

   $0 - Current record (entire line)
   $1 - First field in record
   $n - nth field in record
   NR - Record Number (order in database)
   NF - Number of fields in current record



```
[ murray.saul ] awk '/chevy/ {print $2,$3,$4,$5}' cars.txt
nova 79 60 3000
nova 80 50 3500
impala 65 85 1550
```

Using the awk command to print search results by **field number**.

   For a listing of more variables, please consult your course notes.

8. Issue the following linux pipeline command to display the **model**, **year**, **quantity** and price in the **cars.txt** database for makes of **chevy**:
   ```
   awk '/chevy/ {print $2,$3,$4,$5}' cars.txt | tee awk-2.txt
   ```

```
nova 79 60 3000
nova 80 50 3500
impala 65 85 1550
```

   Notice that a **space** is the delimiter for the fields that appear as standard output.

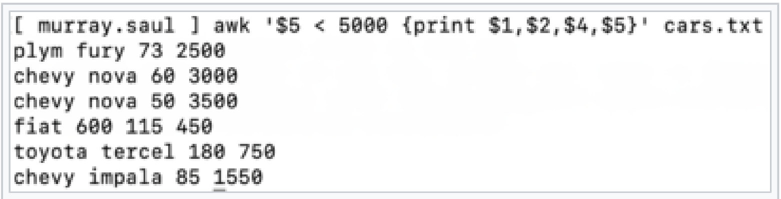
   The **tilde character ~** is used to search for a pattern or display standard output for a particular field.

9. Issue the following linux pipeline command to display all **plymouths** (**plym**) by **model name**, **price** and **quantity**:
   ```
   awk '$1 ~ /plym/ {print $2,$3,$4,$5}' cars.txt | tee awk-3.txt
   ```

```
fury 77 73 2500
```

   You can also use **comparison operators** to specify conditions for processing with matched patterns when using the awk command. Since they are used WITHIN the awk expression, they are not confused with redirection symbols

<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
!=	Not equal



```
[ murray.saul ] awk '$5 < 5000 {print $1,$2,$4,$5}' cars.txt
plym fury 73 2500
chevy nova 60 3000
chevy nova 50 3500
fiat 600 115 450
toyota tercel 180 750
chevy impala 85 1550
```

Using the awk command to display results based on **comparison operators**.

10. Issue the following linux pipeline command to display display the **car make**, **model**, **quantity** and **price** of all vehicles whose **prices are less than $5,000**:
    `awk '$5 < 5000 {print $1,$2,$4,$5}' cars.txt | tee awk-4.txt`

    What do you notice?

```
plym fury 73 2500
chevy nova 60 3000
chevy nova 50 3500
fiat 600 115 450
toyota tercel 180 750
chevy impala 85 1550
```

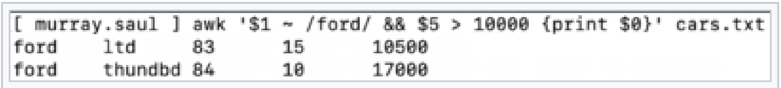
11. Issue the following linux pipeline command to display display **price**, **quantity**, **model** and **car make** of vehicles whose **prices are less than $5,000**:
    `awk '$5 < 5000 {print $5,$4,$2,$1}' cars.txt | tee awk-5.txt`

```
2500 73 fury plym
3000 60 nova chevy
3500 50 nova chevy
450 115 600 fiat
750 180 tercel toyota
1550 85 impala chevy
```

12. Issue the following linux pipeline command to display the **car make**, **year** and **quantity** of cars that **begin** with the **letter 'f'**:
    `awk '$1 ~ /^f/ {print $1,$2,$4}' cars.txt | tee awk-6.txt`

```
ford mustang 45
ford ltd 15
fiat 600 115
ford thundbd 10
ford bronco 25
```

Combined pattern searches can be made by using **compound operator** symbols:



```
[ murray.saul ] awk '$1 ~ /ford/ && $5 > 10000 {print $0}' cars.txt
ford     ltd      83       15       10500
ford     thundbd  84       10       17000
```

Using the awk command to display combined search results based on **compound operators**.

&&	(and)
||	(or)

13. Issue the following linux pipeline command to list all **fords** whose **price is greater than $10,000**:
    `awk '$1 ~ /ford/ && $5 > 10000 {print $0}' cars.txt | tee awk-7.txt`

```
ford     ltd      83    15    10500
ford     thundbd  84    10    17000
```

14. Issue the following linux command (**copy and paste** to save time):
    `wget https://github.com/ULI101/labs/raw/main/cars2.txt`

15. Issue the **cat** command to quickly view the contents of the **cars2.txt** file.

```
plym;fury;77;73;2500
chevy;nova;79;60;3000
ford;mustang;65;45;10000
volvo;gl;78;102;9850
ford;ltd;83;15;10500
chevy;nova;80;50;3500
fiat;600;65;115;450
honda;accord;81;30;6000
ford;thundbd;84;10;17000
toyota;tercel;82;180;750
chevy;impala;65;85;1550
ford;bronco;83;25;9500
```

16. Issue the following linux pipeline command to display the **year** and **quantity** of cars that **begin** with the **letter 'f'** for the **cars2.txt** database:
    `awk '$1 ~ /^f/ {print $2,$4}' cars2.txt | tee awk-8.txt`

    What did you notice? Nothing

The problem is that the **cars2.txt** database separates each field by a semi-colon (;) instead of **TAB**.
Therefore, it does not recognize the second and fourth fields.

You need to issue awk with the -F option to indicate that this file's fields are separated (delimited) by a semi-colorn.

17. Issue the following linux pipeline command to display the **year** and **quantity** of cars that **begin** with the **letter 'f'** for the **cars2.txt** database:

```
awk -F";" '$1 ~ /^f/ {print $2,$4}' cars2.txt | tee awk-9.txt
```

```
mustang 45
ltd 15
600 115
thundbd 10
bronco 25
```

What did you notice this time?

18. Issue the following to run a checking script:

```
~uli101/week10-check-2
```

If you encounter errors, make corrections and **re-run** the checking script until you receive a congratulations message, then you can proceed.

LINUX PRACTICE QUESTIONS

The purpose of this section is to obtain **extra practice** to help with **quizzes**, your **midterm**, and your **final exam**.

Here is a link to the MS Word Document of ALL of the questions displayed below but with extra room to answer on the document to simulate a quiz:

https://github.com/ULI101/labs/raw/main/uli101_week11_practice.docx

Your instructor may take-up these questions during class. It is up to the student to attend classes in order to obtain the answers to the following questions. Your instructor will NOT provide these answers in any other form (eg. e-mail, etc).

Review Questions:

Part A: Display Results from Using the sed Utility

Note the contents from the following tab-delimited file called **~murray.saul/uli101/stuff.txt**: (this file pathname exists for checking your work)

```
Line one.
This is the second line.
This is the third.
This is line four.
Five.
Line six follows
Followed by 7
Now line 8
and line nine
Finally, line 10
```

Write the results of each of the following Linux commands for the above-mentioned file:

1. **`sed -n '3,6 p' ~murray.saul/uli101/stuff.txt`**

2. **`sed '4 q' ~murray.saul/uli101/stuff.txt`**

3. **`sed '/the/ d' ~murray.saul/uli101/stuff.txt`**

4. **`sed 's/line/NUMBER/g' ~murray.saul/uli101/stuff.txt`**

Part B: Writing Linux Commands Using the sed Utility

Write a single Linux command to perform the specified tasks for each of the following questions.

1. Write a Linux sed command to display only lines 5 to 9 for the file: **~murray.saul/uli101/stuff.txt**

2. Write a Linux sed command to display only lines the begin the pattern “and” for the file: **~murray.saul/uli101/stuff.txt**

3. Write a Linux sed command to display only lines that end with a digit for the file: **~murray.saul/uli101/stuff.txt**

4. Write a Linux sed command to save lines that match the pattern “line” (upper or lowercase) for the file: **~murray.saul/uli101/stuff.txt** and save results (overwriting previous contents) to: **~/results.txt**

Part C: Writing Linux Commands Using the awk Utility

Note the contents from the following tab-delimited file called **~murray.saul/uli101/stuff.txt**: (this file pathname exists for checking your work)

```
Line one.
This is the second line.
This is the third.
This is line four.
Five.
Line six follows
Followed by 7
Now line 8
and line nine
Finally, line 10
```

Write the results of each of the following Linux commands for the above-mentioned file:

1. **`awk ‘NR == 3 {print}’ ~murray.saul/uli101/stuff.txt`**

2. **`awk ‘NR >= 2 && NR <= 5 {print}’ ~murray.saul/uli101/stuff.txt`**

3. `awk '$1 ~ /This/ {print $2}' ~murray.saul/uli101/stuff.txt`

4. `awk '$1 ~ /This/ {print $3,$2}' ~murray.saul/uli101/stuff.txt`

Part D: Writing Linux Commands Using the awk Utility

Write a single Linux command to perform the specified tasks for each of the following questions.

1. Write a Linux awk command to display all records for the file: **~/cars** whose fifth field is greater than 10000.
2. Write a Linux awk command to display the first and fourth fields for the file: **~/cars** whose fifth field begins with a number.
3. Write a Linux awk command to display the second and third fields for the file: **~/cars** for records that match the pattern “chevy”.
4. Write a Linux awk command to display the first and second fields for all the records contained in the file: **~/cars**

Category: ULI101