CSc 3320: Systems Programming

Fall 2021

Homework

2: Total points 100

Submission instructions:

1. Create a Google doc for each homework assignment submission.
2. Start your responses from page 2 of the document and copy these instructions on page 1.
3. Fill in your name, campus ID and panther # in the fields provided. If this information is missing in your document TWO POINTS WILL BE DEDUCTED per submission.
4. Keep this page 1 intact on all your submissions. If this *submissions instructions* page is missing in your submission TWO POINTS WILL BE DEDUCTED per submission.
5. Each homework will typically have 2-3 PARTS, where each PART focuses on specific topic(s).
6. Start your responses to each PART on a new page.
7. If you are being asked to write code copy the code into a separate txt file and submit that as well.
8. If you are being asked to test code or run specific commands or scripts, provide the evidence of your outputs through a screenshot and copy the same into the document.
9. Upon completion, download a .PDF version of the document and submit the same.

Full Name: Robert Tognoni

Campus ID: rtognoni1

Panther #: 002-50-0041

Note: all screenshots attached separately and labeled by question number in format "HW2_Q# firstnameLastname.pdf"

PART 1 (2.5 points each): 10pts

1. What are the differences among ***grep***, ***egrep*** *and* ***fgrep***? Describe using an example.

   Answer:

   grep: Pattern match for words. Does not recognize extended regex characters.

   Example: grep 'wombat' file.txt >> will search for lines with zero or more instances of wombat.*

   egrep: Pattern matching with extended regex symbols such as +, ?, !, &, (,) etc.

   Example: egrep '(wombat)+' file.txt >> will search for lines with one or more instances of wombat.

   fgrep: Pattern is interpreted as a list of string literals to be matched. Treats regex symbols as exact characters. Considered a faster search

   Example: fgrep '^a?' file.txt >> will search for an exact match to the characters ^a? In the text file

2. Which utility can be used to compress and decompress files? And how to compress multiple files into a single file? Please provide one example for it.

   Answer:

   Command: *tar*

   To compress multiple files: *tar -cf file1.txt file2.py*

3. Which utility (or utilities) can break a line into multiple fields by defining a separator? What is the default separator? How to define a separator manually in the command line? Please provide one example for defining the separator for each utility.

    Answer: sort and awk

    sort: The default separator is {space}. To define a different separator use the command -tx where x is the separator character.
    Example: sort -t; -k2 stuff.txt >> will sort based on field 2 that is delimited by the ; symbol.

    awk can also change the separator with the -Fx option where x is the delimiting character.
    Example: awk -F: '{print NF, $1}' stuff.txt

4. What does the ***sort*** command do? What are the different possible fields? Explain using an example.

    Answer:

    Sort will sort a file or text stream in asc or desc order based on one or more fields.
    Fields are determined by a preset delimiter (defaulted to space) that separates text into 'columns' per line.

    Example:
    cat stuff.txt >> (display contents)
    Happles:Kapples:423
    Apples:Grapples:721
    Rapples:Clapples:123

sort -t: -k2 stuff.txt >> (will set delimiter to ':' and sort by column2)
Rapples:Clapples:123
Apples:Grapples:721
Happles:Kapples:423

sort -t: -k3 stuff.txt >> (will set delimiter to ':' and sort by column3)
Rapples:Clapples:123
Happles:Kapples:423
Apples:Grapples:721

Part IIa (5 points each): 25pts

5. What is the output of the following sequence of bash commands: **echo 'Hello World' | sed 's/$/!!!/g'**

   Answer:
   Output: Hello World!!!
   Description: pipe 'Hello World' into sed and replace end of line with '!!!'

6. What is the output for each of these awk script commands?

   -- 1 <= NF { print $5 }
   Answer: *Print from field 5 if the number of fields is greater than or equal to 1.*

   -- NR >= 1 && NR >= 5 { print $1 }
   Answer: *Print field 1 if the line number is 5 or greater.*
   {If this question was meant to be NR >=1 && NR <=5 it would mean "print lines between 1 and 5 inclusive".}

   -- 1,5 { print $0 }
   Answer: Print from field 0 (entire line)

   -- {print $1 }
   Answer: Print from field 1 for all lines.

7. What is the output of the following command line:
   echo good | sed '/Good/d'

   Output: *good*
   Description: *pipe 'good' into sed and delete lines matching 'Good'. In this case, nothing happens as 'good' does not match 'Good'.*

8. Which **awk** script outputs all the lines where a plus sign + appears at the end of line?

    Answer:

    /\+$/ { print }

    Inline: awk '/\+$/ { print }' stuff.txt

9. What is the command to delete only the first 5 lines in a file "foo"? Which command deletes only the last 5 lines?

    Answer:

    a) sed '1,5 d' foo
    b) head -n -5 foo

Part IIb (10pts each): 50pts

Describe the function (5pts) and output (5pts) of the following commands.

9. **$ cat float**

Wish I was floating in blue across the sky, my imagination is strong,
And I often visit the days
When everything seemed so clear.
Now I wonder what I'm doing here at all...

$ cat h1.awk

```
NR>2 && NR<4{print NR ":" $0
```

$ awk '/.*ing/ {print NR ":" $1}' float

Answer:

*Match line to .*ing regex (0 or more chars before 'ing') and print the line number : (field 1) (based on space separation).*

Output:

```
>>1:Wish
>>3:When
>>4:Now
```

10. As the next command following question 9,

$ awk -f h1.awk float

Answer:

Due to lack of trailing '}' symbol in h1.awk this produces a syntax error however if this is fixed:

The command runs the h1.awk script on the float text file. h1.awk will print any line that is greater than 2 and less than 4, so it prints line 3. Note that field $0 refers to the entire line.

Output:

```
>>3:When everything seemed so clear
```

11.

$ **cat h2.awk**

```
BEGIN { print    "Start to scan file" }
{print  $1      ","  $NF}

END {print     "END-" , FILENAME }
```

$ awk -f h2.awk float

Answer:

Script will first print “Start to scan file” before operation.

Then for each line it will print the first field and the last field of the line. $NF is the number of fields for the line which equates to the last field when passed as the field variable.

Finally, the script will print “END-” and the name of the file which is float.

Output:

```
>>Start to scan file
>>Wish,Strong,
>>And,days
>>When,clear
>>Now,all…
>>END- float
```

12.

sed 's/\s/\t/g'   float

Answer:

Replace all spaces with tabs on all lines in the float file.

Output: (formatting is kind of off)

Wish	**I**	**was**	**floating**	**in**	**blue**	**across**	**the**	**sky,**
m y	**imagination**	**is**	**strong,**					
And	**I**	**often**	**visit the**	**days**				
When	**everything**	**seemed**	**so**	**clear**				
Now	**I**	**wonder**	**what**	**I'm**	**doing**	**here**	**at**	**all...**

13.

$ ls *.awk| awk '{print "grep --color 'BEGIN' " $1 }' |sh *(Notes:* ***sh file*** *runs file as a shell script . $1 should be the output of ‘* ls *.awk ‘ in this case, not the 1st field)*

Answer:

1) Make a list of files that end in .awk
2) Pipe into an awk inline script that prints “grep --color ‘BEGIN’” and filed 1 which is the full file name of the awk files in this case.
3) Pipe previous into sh command and execute.
4) Grep will execute the commands provided and highlight any lines that contain ‘BEGIN’ in whatever the set color is (red).

Output:

Depends on the files in the working directory. But for example:

```
BEGIN { print NF "Start to scan file"}
```

14.

```
$ mkdir  test     test/test1  test/test2
$cat>test/testt.txt
This is a test file ^D
$ cd  test
$ ls  -l . | grep '^d' | awk '{print "cp -r " $NF " " $NF ".bak"}' | sh
```

Answer:

1) Create the dir test and subdirs test1 and test2 within it.
2) Create the file test.txt in the test directory with the given string.
3) Move CWD to the test directory.
4) List all files with access information
5) Pipe this ls output into grep and search for lines that start with ‘d’
6) Pipe grep output into awk command that will write the lines “cp -r {file name} {filename}.bak
7) Pipe awk output into sh to run as shell command.
8) test1 and test2 commands are copied and renamed to test1.bak and test2.bak files.

Output:

There is no visible output, but the list of files in /test will be as follows:

test 1 test1.bak test2 test2.bak test.txt

Part III Programming: 15pts

15. Sort all the files in your class working directory (or your home directory) as per the following requirements:

a. A copy of each file in that folder must be made. Append the string "_copy" to the name of the file

command: `ls -1 -p | grep -v  / | awk -F. '{print "cp " $0 " " $1 "_copy." $2}'|sh`
description:

1. ls -1 -p will list files 1 per line with '/' included on directories
2. grep -v / will list files that do not include a '/' (to get a list of files only)
3. awk -F. '{print "cp " $0 " " $1 "_copy." $2}' sets up the command strings to make the copies. To insert _copy before the file type had to split by '.' and use fields $1 and $2 to reconstruct
4. sh runs each given line as a shell command.

b. The duplicate (copied) files must be in separate directories with each directory specifying the type of the file (e.g. txt files in directory named txtfiles, pdf files in directory named pdffiles etc).

commands:

1) `ls -1 | awk -F. '/_copy/ {print "mkdir " $2 "_files"}' | sort -u | sh`
2) `ls -1 | awk -b -F. '/_copy/ {print "mv "$0 " " $2 "_files/"}'| sh`

Description:

1. List files one per line.
2. Set awk delimiter to '.'
3. Filter for lines with '_copy'
4. Print strings mkdir {file type}_files *eg. txt_files*
5. Sort only unique entries to remove duplicate files
6. Pipe into sh and run each line as shell command to create folders
7. Steps 1-3 as listed
8. Print strings mv {full file name} {file type}_files/
9. Pipe into sh and run each line as shell command to move

each file in the list.

c. The files in each directory must be sorted in chronological order of months.

Command: ls -1Rltr
Description: sorts listed files in chronological order (oldest to newest) and displays the date info. Applied to all dirs and subdirs.

d. An archive file (.tar) of each directory must be made. The .tar files must be sorted by name in ascending order.
Command:
1) ls -1p | grep / | awk -F/ '{print "tar -cf " $1 ".tar " $1}'|sh
2) ls | grep .tar | sort
Description:
1. Send list of files to grep and filter out directories based on '/' character. -p includes '/' on directories so we can filter them from files.
2. Create common strings for tar command using name of file (field $1 delimited by /) and .tar
3. Pipe into sh to run commands
4. List .tar folders in ascending order as requested. Though it looks like the system already defaults to this method of sorting anyway.

e. An archive file of all the .tar archive files must be made and be available in your home directory.

Note: I originally had a more complicated way of doing this that involved moving the .tar files to a specific folder first. My screenshot reflects adjustments I had to make to do it by this simpler method after unpacking the files into a separate directory.

Command:
1. tar -cf ~/tar_archive.tar *.tar
2. tar -tvf tar_archive.tar

Description:
1. Create an archive of all .tar files in the home directory

2. List files inside the created tar (optional)

As an output, show your screen shots for each step or a single screenshot that will cover the outputs from all the steps.