CSc 3320: Systems Programming

Spring 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: Rafid H. Shaon

Campus ID: rshaon1

Panther #: 002-49-7367

PART 1 (2.5 points each): 10pts

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

grep

grep is an acronym that stands for "Global Regular Expressions Print". grep is a program which scans a specified file or files line by line, returning lines that contain a pattern. A pattern is an expression that specifies a set of strings by interpreting characters as meta-characters. For example the asterisk meta character (*) is interpreted as meaning "zero or more of the preceding element". This enables users to type a short series of characters and metacharacters into a grep command to have the computer show us what lines in which files match.

The standard grep command looks like:

```
grep <flags> '<regular expression>' <filename>
```

grep prints the search results to the screen (stdout) and returns the following exit values:

```
0 A match was found. 1 No match was found. >1 A syntax error was found or a
file was inaccessible (even if matches were found).
```

Some common flags are: -c for counting the number of successful matches and not printing the actual matches, -i to make the search case insensitive, -n to print the line number before each match printout, -v to take the complement of the regular expression (i.e. return the lines which don't match), and -l to print the file names of files with lines which match the expression.

egrep

egrep is an acronym that stands for "Extended Global Regular Expressions Print".
The 'E' in egrep means treat the pattern as a regular expression. "Extended Regular Expressions" abbreviated 'ERE' is enabled in egrep. egrep (which is the same as `grep -E`) treats `+`, `?`, `|`, `(`, and `)` as meta-characters.

In basic regular expressions (with grep), the meta-characters `?`, `+`, `{`, `|`, `(`, and `)` lose their special meaning. If you want grep to treat these characters as meta-characters, escape them `\?`, `\+`, `\{`, `\|`, `\(`, and `\)`.

For example, here grep uses basic regular expressions where the plus is treated literally, any line with a plus in it is returned.

```
grep "+" myfile.txt
```

egrep on the other hand treats the "+" as a meta character and returns every line because plus is interpreted as "one or more times".

```
egrep "+" myfile.txt
```

Here every line is returned because the + was treated by egrep as a meta character. normal grep would have searched only for lines with a literal +.

fgrep

fgrep is an acronym that stands for "Fixed-string Global Regular Expressions Print".
fgrep (which is the same as grep -F) is fixed or fast grep and behaves as grep but does NOT recognize any regular expression meta-characters as being special. The search will complete faster because it only processes a simple string rather than a complex pattern.
For example, if I wanted to search my .bash_profile for a literal dot (.) then using grep would be difficult because I would have to escape the dot because dot is a meta character that means 'wild-card, any single character':

```
grep "." myfile.txt
```

The above command returns every line of myfile.txt. Do this instead:

```
fgrep "." myfile.txt
```

Then only the lines that have a literal '.' in them are returned. fgrep helps us not bother escaping our meta characters.

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.

tar:

tar creates the archives and extracts the archives too.

Modes:

-c create an archive.

-x extract an archive.

tar archives without compression, then it can compress with gzip, the result file has a .tar.gz extension.

COMPRESS SINGLE DIRECTORY OR A SINGLE FILE:

For example there is a directory ‘pictures’ in the current directory and would like to save as images.

Then the command to compress is

$ tar -czvf images.tar.gz pictures

czvf means:

-c create an archive

-z compress the archive using gzip

-v display progress in terminal

-f to specify filename

For example there is a directory ‘pictures’ at /user/name/pictures and would like to save as images.

Then the command to compress is

$ tar -czvf images.tar.gz /user/name/pictures

COMPRESS MULTIPLE DIRECTORIES OR MULTIPLE FILES:

For example there is a directory

‘pictures’ at /user/name/pictures

‘music’ at /user/name/music

‘videos’ at /user/name/videos

would like to save as media.

Then the command to compress is

$ tar -czvf media.tar.gz /user/name/pictures /user/name/music /user/name/videos

(NOTE: each path is separated by space)

DECOMPRESS ARCHIVE:

For example there is an archive media.tar.gz would like to decompress to current directory

Then the command to decompress is

$ tar -cxvf media.tar.gz

(NOTE: mode set to x, for extraction)

To decompress to /user/name/files directory, then the command to decompress is

$ tar -cxvf media.tar.gz –C /user/name/files

(NOTE: -C allows to extract into specified path)

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.

awk

If you want to use awk, the way to supply the delimiter is either through the -F argument or as a FS= postfix:

awk -F '\t' '{ print $2 }' infile

Or:

awk '{ print $2 }' FS='\t' infile

Output in all cases:

"Here's the field of the text, also contains comma"

"Here's the field of the text, also contains comma"

"Here's the field of the text, also contains comma"

"Here's the field of the text, also contains comma"

Quote delimiter

If the double-quotes in the file are consistent, i.e. no embedded double-quotes in fields, you could use them as the delimiter and avoid having them in the output, e.g.:

cut

cut -d\" -f4 infile

awk

awk -F\" '{ print $4 }' infile

Output in both cases:

Here's the field of the text, also contains comma

Here's the field of the text, also contains comma

Here's the field of the text, also contains comma

Here's the field of the text, also contains comma

To specific the tab as a delimiter

$ awk -F '\t' '{print $2}' file.csv

To take away the unwanted "

$ awk -F '\t' '{print $2}' file.csv | sed 's/"//g'

Other option using awk -F

$ awk -F '"' '{print $4}' file.csv

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

sort command is the one used for sorting the input file and arranging the records in some mentioned order. There are thus different options to sort the data with the command on Linux. Sorting can be done numerically, alphabetically, reverse, and other mechanisms.

sort -0 option to write the output to an output file

sort -r option to sort the file in reverse order

sort -n is done to sort the file numerically

sort -nr is done to sort file numerically in reverse

sort -k is used for sorting the file which consists a table and k is used to denote the table

There are also:

-u option for sort and remove duplicates at the same time

-M option for sorting records by month

For example:

input.txt contains:

anil

piyush

zebra

cat

dog

Command:

sort -r input.txt

will sort the output as:

zebra

piyush

dog

cat

anil

Part IIa (5 points each): 25pts

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

Hello World!!!

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

-- 1 <= NF { print $5 }

awk '1 <= NF { print $5 }'

Would print fifth field (column) if 1 is less than or equal to number of fields (NF). If NF is less than 5 then it won't print anything.

-- NR >= 1 && NR >= 5 { print $1 }

awk 'NR >= 1 && NR >= 5 { print $1}'

Would print the first column for all the rows greater than or equal to 1 AND greater than or equal to 5.

-- 1,5 { print $0 }

awk '1,5 { print $0 }'

Would print all the lines in a file as 1,5 would evaluate to TRUE.

-- {print $1 }

awk '{ print $1 }'

Would print the first column in each line. We can provide the field separator using -F option.
The default FS is space.

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

   good

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

   awk '$0 ~ /+$/ { print $0 }'

   OR

   awk '/+$/ { print $0 }'

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

   Commands for deleting the first 5 lines from the file foo:
   Command 1: sed -i '1,5d' foo

   Command 2: tail -n +5 foo > foo.tmp ; mv foo.tmp foo

   Command 3: awk 'NR > 5 { print $0 }' foo > foo.tmp ; mv foo.tmp foo

   Commands for deleting the last 5 lines from the file foo:
   Command 1: tac foo | sed "1,5d" | tac > foo.tmp ; mv foo.tmp foo

   Command 2: head -n -5 foo > foo.tmp ; mv foo.tmp 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

awk '/.*ing/ {print NR ":" $1}' float
Would print the record number, a colon and the first field for the lines it finds 'ing' in any of the strings.

The output:
1:Wish
3:When
4:Now

Explanation:
Anything inside // is used to find the string(s) in the file.

Here we use a regular expression .*ing which means any string containing 'ing' anywhere.

The script finds 3 such lines and prints them along with their record numbers.

10. As the next command following question 9,

$ awk -f h1.awk float

awk -f h1.awk float
This would print the record number, a colon and the third line from the file float.

The output:
3:When everything seemed so clear.

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

The awk will act on record numbers greater than 2 and less than 4 (which is 3). The print would print the record number followed by a colon and then the entire line ($0).

11.

```
$ cat h2.awk
BEGIN { print    "Start to scan file" }
{print  $1      ","  $NF}

END {print     "END-" , FILENAME }
$ awk -f h2.awk float
```

awk -f h2.awk float
This would print 'Start to scan file' in the beginning, then for each record, it would print the first column followed by a comma and the last column followed by the input filename.

The output:
Start to scan file
Wish,is
strong,,days
When,clear.
Now,all...
END- float

Explanation:
awk script would first execute the commands in the BEGIN section once per file, then it would execute the commands in the executable section once for each record, and then finally the commands in the END section.

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

sed 's/\s/\t/g' float
This would replace space with tab for all occurrences of space.

The output:

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**	**wonderwhat**	**I'm**	**doing**	**here**	**at**	**all...**

Explanation:
Here the sed will find space (\s, escaped) with tab (\t) and the flag g would do it for all the occurrences.

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)*

ls *.awk | awk '{print "grep --color 'BEGIN' "$1}' | sh

Explanation:
The first command ls *.awk would list all the .awk files in the current directory (h1.awk and h2.awk in this case).
The awk would then build a grep command to list all the lines containing BEGIN keyword.
The final sh would then run the commands generated.

The intermediate command generated would be:
grep --color BEGIN h1.awk
grep --color BEGIN h2.awk

The output:
BEGIN { print "Start to scan file" }

~ The keyword BEGIN would be printed in red as we have used --color option of grep.

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
```

mkdir test test/test1 test/test2
This would create a directory with the name 'test' and inside it two more directories 'test1' and 'test2'.

cat>test/testt.txt
This is a test file^D

This would create a file testt.txt inside test directory with the contents 'This is a test file'.

cd test
would enter into the test directory.

ls -l . | grep '^d' | awk '{print "cp -r " $NF " "$NF".bak"}' | sh

Explanation:
The 'ls -l .' would long list the files in the test directory. The grep would then find lines starting with d (i.e. the directories). awk will then recursively copy the directories with the respective directories by appending the directory names with .bak.

The output:
ls -p
test1/ test1.bak/ test2/ test2.bak/ testt.txt

~ The -p option puts slash after each directory listed.

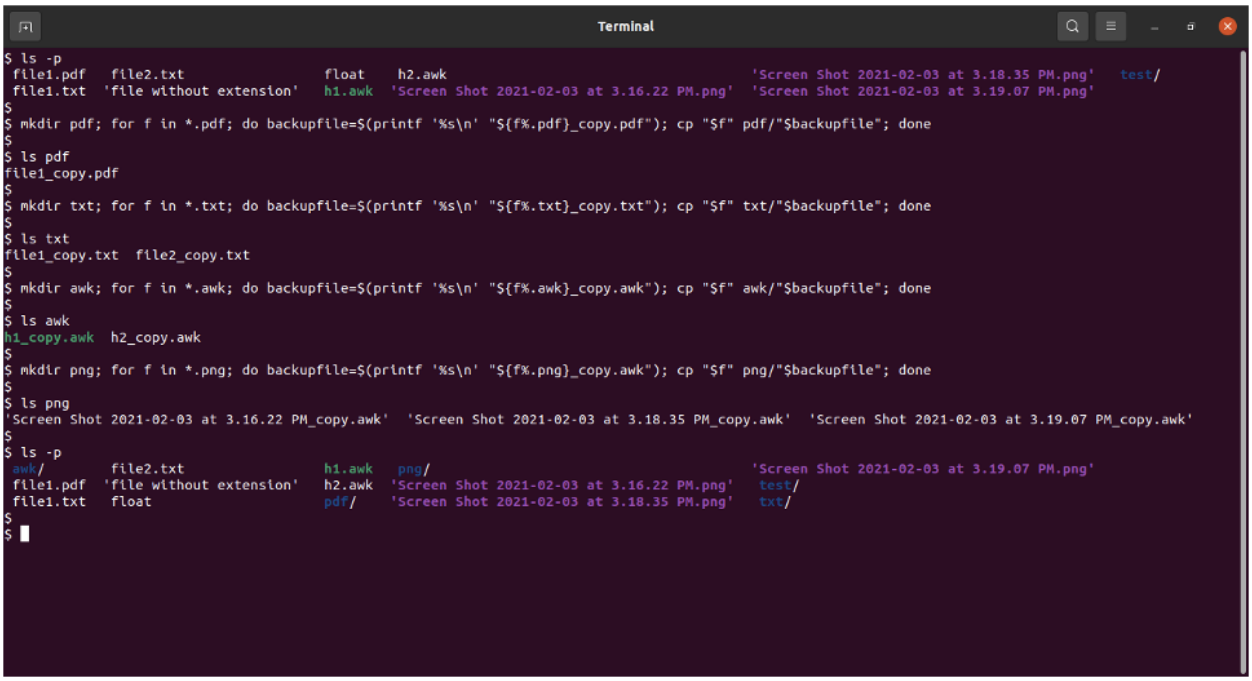
Part III Programming: 15pts

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

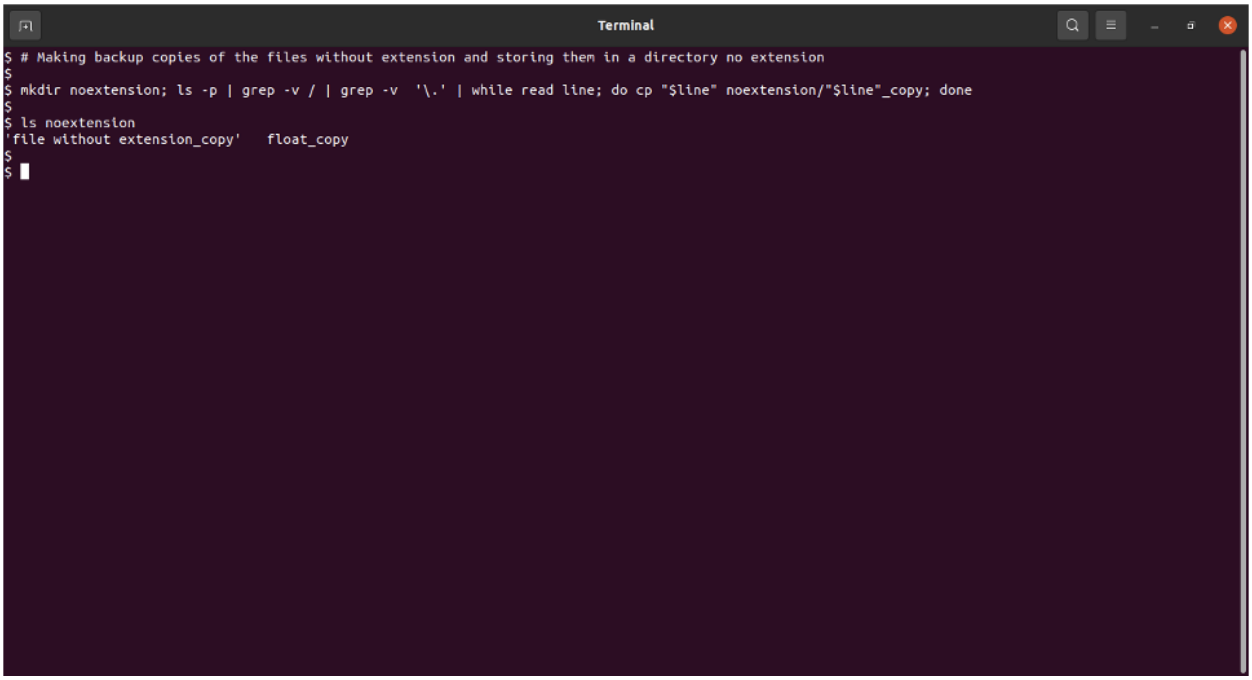
a. A copy of each file in that folder must be made. Append the string "_copy" to the name of the file
b. The duplicate (copied) files must be in separate directories with each directory specifying the type of the file (e.g. txt files in directory named txtfiles, pdf files in directory named pdffiles etc).
c. The files in each directory must be sorted in chronological order of months.
d. An archive file (.tar) of each directory must be made. The .tar files must be sorted by name in ascending order.
e. An archive file of all the .tar archive files must be made and be available in your home directory.

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

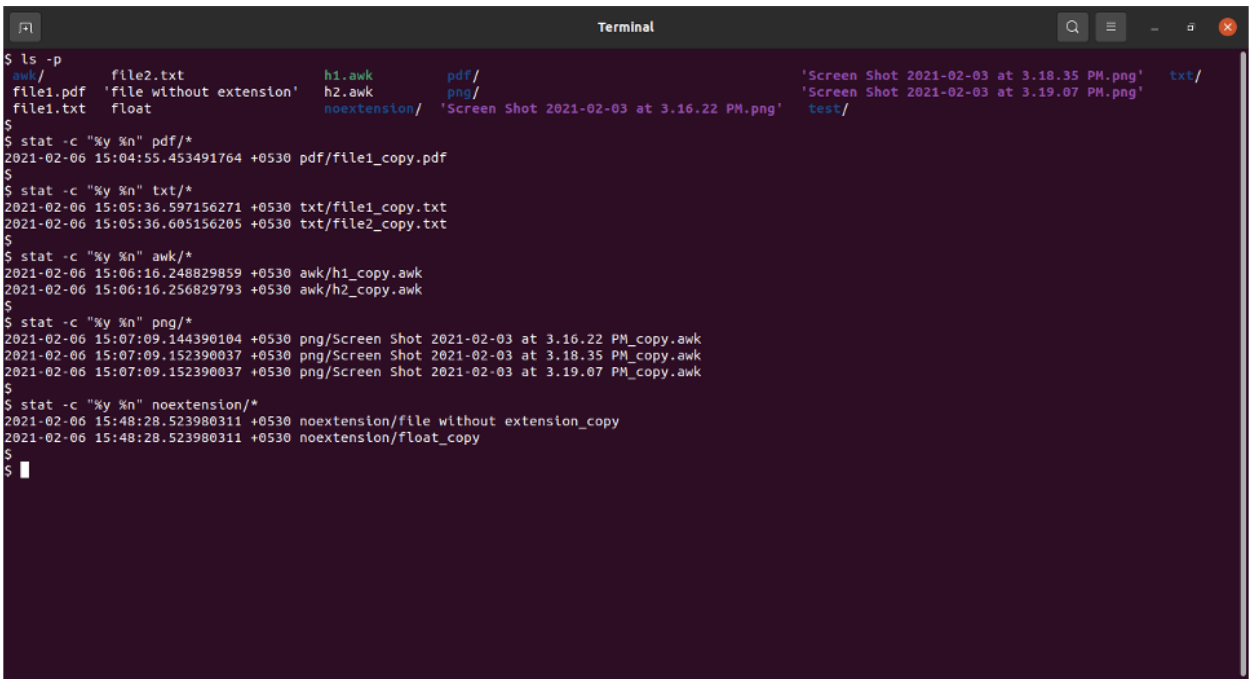
Making backup copies of the files with extension and storing them in respective directories created according to their extensions.



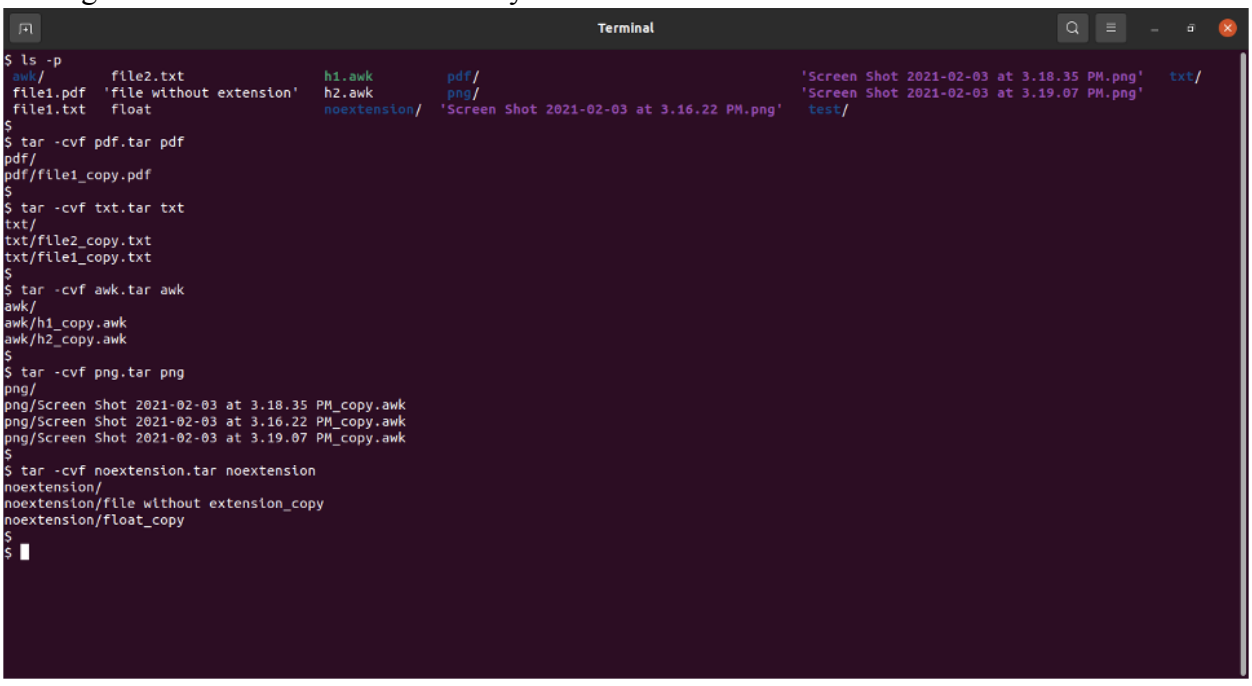
Making backup copies of the files without extension and storing them in a directory noextension.



Displaying files in each directory sorted by month in a chronological order.

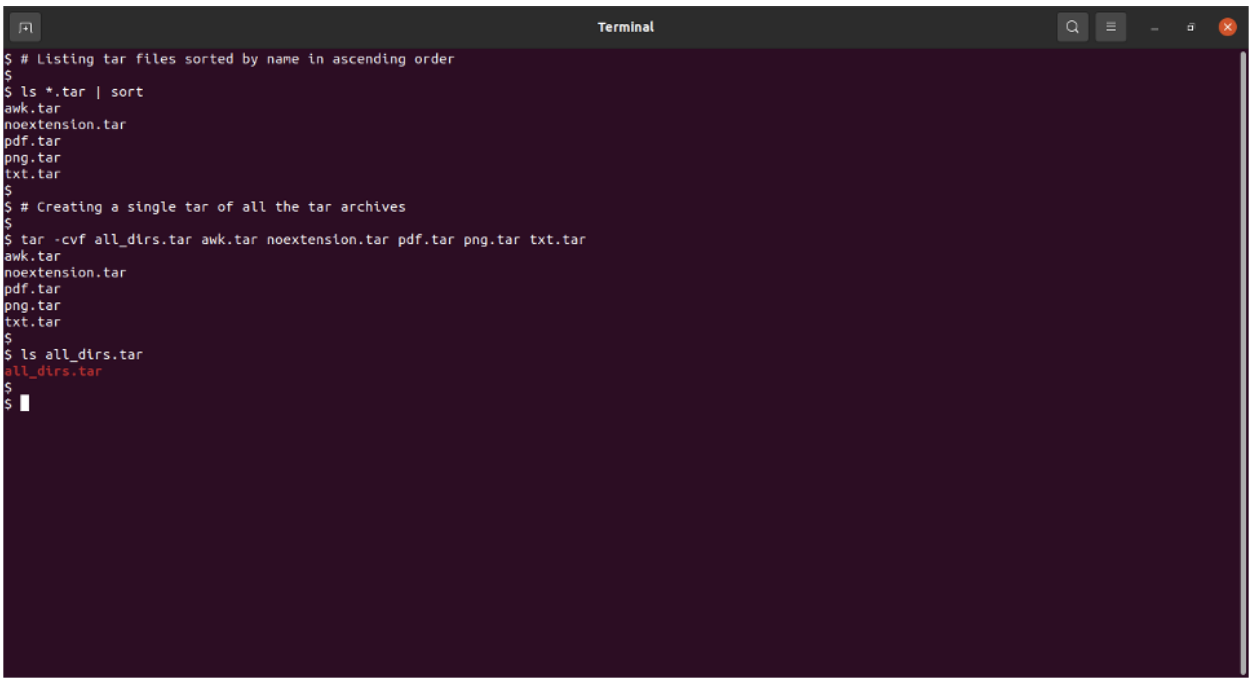


Making the tar archives of each directory.



Listing tar archive files sorted by name in ascending order and creating a single tar of all the tar

archives


Terminal
$ # Listing tar files sorted by name in ascending order
$
$ ls *.tar | sort
awk.tar
noextension.tar
pdf.tar
png.tar
txt.tar
$
$ # Creating a single tar of all the tar archives
$
$ tar -cvf all_dirs.tar awk.tar noextension.tar pdf.tar png.tar txt.tar
awk.tar
noextension.tar
pdf.tar
png.tar
txt.tar
$
$ ls all_dirs.tar
all_dirs.tar
$
$