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 *submission's 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: Vivian Do

Campus ID: vdo10

Panther #: 002486640

Due Oct 01, 2021

PART I (2.5 points each): 10pts

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

grep: “Global Regular Expressions Print”

- grep is a program that scans a specific file or files line by line and returns lines with patterns. A pattern is an expression that identifies a set of strings by understanding characters as meta-characters.
- For instance, the asterisk meta character * is understood as “zero or more of the preceding element”. This allows users to type characters and meta-characters into a grep command. This will have the computer show what lines in which file will match.
- standard grep command:
    - `grep <flags> ‘<regular expression>’ <filename>`
- grep prints the search results to the screen (stdout) and returns this exit value:
    - `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)`
- Common Flags:
    - *-c*: counts the number of matches but doesn’t print actual matches
    - *-i*: makes searches case insensitive
    - *-n*: prints line number before each match printout
    - *-l*: prints file names of files with lines matching the expression

egrep: “Extended Global Regular Expressions Print”

- egrep handles patterns as regular expressions. It also handles `+`, `?` , `|`, `(`, and `)` as meta-characters.
- Meta-characters lose their special meaning in grep. In order for grep to treat these characters as meta-characters, put a backslash before the character `\+`, `\?`, `\|`.
- grep using basic regular expression where the plus + is treated literally:
    - `grep “+” file.txt`
        - this returns any line with a plus +
- egrep treating the plus + as a meta-character and returns every line:
    - `egrep “+” file.txt`
- every line is returned since the plus + was treated by egrep as a meta-character; normal grep searches for only the lines with a plus +

fgrep: “Fixed-string Global Regular Expressions Print”

- fgrep, or *grep -F*, is a fixed/fast grep that behaves as a grep but doesn't recognize any regular expression meta-characters as anything special. This search completes faster because it only processes simple strings in comparison to complex patterns.
- When searching for a .bash_profile for a literal dot . using grep would be difficult. The dot . needs to be escaped because it's a meta-character meaning "wild-card, any single character'.
    - `grep "." file.txt`
- This command returns every line of the file.txt
    - `fgrep "." file.txt`
- Only the lines with a literal dot . in this command will be returned.
- fgrep helps by not bothering to escape the 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 and extracts archives
    - tar archives without compression and then compresses with gzip. The file will then have an extension of *.tar.gzip*

- Modes:
    - `-c`: creates an archive
    - `-x`: extracts an archive

- Compress Single Directory/File:
    - Let there be a directory called *pictures* in the current directory that wants to save images. Below is the command:
    - `$ tar -czvf images.tar.gz pictures`
        - *-c*: creates an archive
        - *-z*: compress the archive using gzip
        - *-v*: display progress in terminal
        - *-f*: specifying file name
    - Let there be a directory called *pictures* in */user/name/pictures* that wants to save images. Below is the command:
    - `$ tar -czvf images.tar.gz /user/name/pictures`

- Compress Multiple Directories/Files:
  - Let there be a directory called “pictures” at /user/name/pictures, “music” at /user/name/music, and “videos” at /user/name/videos and it would like to save as media. Below is the command:
  - `$ tar -czvf media.tar.gz /user/name/pictures /user/name/music /user/name/videos`
    - Each path is separated by a space.

- Decompress Archive:
  - Let there be an existing archive *media.tar.gz* that wants to decompress to the current directory. Below is the command:
  - `$ tar -cxvf media.tar.gz`
    - Mode is set to *x* for extraction
  - To decompress to */user/name/files* directory, this is the command to decompress:
  - `$ tar -cxvf media.tar.gz -C /user/name/files`
    - *-C* allows extracting to specific paths

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
- When using awk, the way to supply the delimiter is either through *-F* or a *FS=* postfix:
  - `awk -F ‘\t’ { print $2 }’ file` OR `awk ‘{ print $2 } FS=’\t’ file`
  - Output in all cases:

        “Testing one, two, three”
        “Testing one, two, three”
        “Testing one, two, three”
        “Testing one, two three”
- Quote Delimiter: if the double quote in the file are consistent, they can be used as a delimiter and can avoid the output using *cut*

        - `cut`
        - `cut -d\” -f4 file`
        - `awk`
        - `awk -F\” ‘{ print $4 }’ file`
        - Output in all cases:
            - Testing one, two, three
            - Testing one, two, three
            - Testing one, two, three
            - Testing one, two, three
    - specifying tab as a delimiter:
        - `$ awk -F ‘\t’ {print $2}’ file.csv`
    - taking away the unwanted quotations “:
        - `$ awk -F ‘\t’ {print $2}’ file.csv | sed ‘s/”//g’`
    - other ways to use awk -F
        - `$ 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 can sort the input file and arrange the records in a specific order. There are different options to sort the data in Linux. Sorting can be done numerically, alphabetically, in reverse, and many other ways.
    - *sort -0*: writes the output to an output file
    - *sort -r*: sorts the file in reverse
    - *sort -n*: sorts the file numerically
    - *sort -nr*: sorts the file numerically in reverse
- Other Options:
    - *sort -k*: sorts the file with a table; k is used to denote the table
    - *sort -u*: sorting and removing duplicated at the same time
    - *sort -M*: sorting records by month
- Example:
- Let the text file *color.txt* have

    red
    yellow
    green

blue

purple

- Command: `sort -r color.txt`
- Sorted Output:

yellow

red

purple

green

blue

Part II a (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 }'`
    - prints the fifth field/column if 1 is less than or equal to the number of fields (NF). If the number of fields is less than 5, then nothing will print

-- NR >= 1 && NR >= 5 { print $1 }
- `awk 'NR >= 1 && NR >= 5 { print $1 }'`
    - prints 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 }'`
    - prints all the lines in a file as 1,5; evaluates TRUE

-- {print $1 }
- `awk '{ print $1 }'`
    - prints the first column in each line; using *-F* will provide a field separator; the default FS is space

7. What is the output of the 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?

- Command to delete only the first 5 lines in a file "foo":
    - `sed -i '1,5d' foo`

  - `tail -n +5 foo > foo.temp ; mv foo.temp foo`
  - `awk ‘NR > 5 { print $5 }‘ foo > foo.temp ; mv foo.temp foo`

- <u>Command to delete only the last 5 lines:</u>
  - `tac foo | sed “1,5d” | tac > foo.temp ; mv foo.temp foo`
  - `head -n -5 foo > foo.temp ; mv foo foo.temp`

Part II b (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`
- prints the record number, a colon, and the first field for the lines finding "ing" in any of the strings
- Output:

  1:Wish
  2:When
  3:Now
- Explanation: Anything inside the double backslashes // is used to find the string(s) in the file. The expression . **ing* will find any string with "ing" anywhere in the file. The script will find three lines and print them with their record numbers.

10. As the next command following question 9,

$ awk -f h1.awk float

- `awk -f h1.awk float`
- prints the record number, a colon and the third line form the file *float*
- Output:

  3:When everything seemed so clear.
- Explanation: The command *NR>2 && NR<4{print NR ":" $0}* makes *awk* act on record numbers greater than 2 and less than 4 (the number is 3). The record number, a colon and the entire line($0) will print.

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`
- prints "Start to scan file" in the beginning, then for each record it would print the first column, followed by a comma, and another comma followed by the filename
- Output:

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

- Explanation: The *awk* script first executes the commands in the *BEGIN* section once per file, then executes 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`
- replaces space with tab for all occurrences of space.
- 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 wonder                what    I'm     doing   here    at
all...
```

- Explanation: The *sed* will find space (\s) with tab (\t) and the flag *g* would replace them for all 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 command *ls *awk* lists all the *.awk* files in the current directory (*h1.awk* and *h2.awk*). The *awk* then builds a grep command to list all the lines with the *BEGIN* keyword. The *sh* then runs the generated commands.
- the generated commands:

```
grep --color BEGIN h1.awk
grep --color BEGIN h2.awk
```

- Output:

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

  - (the keyword BEGIN would be printed in red since we 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 creates a directory named *test* with two more directories inside it named *test1* and *test2*
- `cat>test/testt.txt`
- `This is a test file^D`
    - This creates the *test.txt* file inside the test directory with the contents "This is a test file".
- `cd test` (is entered into *test* directory)
- `ls -l . | grep '^d' | awk '{print "cp -r " $NF " "$NF".bak"}' | sh`
- Explanation: The *ls -l* lists the files in the *test* directory. The *grep* then finds the lines starting with *d* (the directories). *awk* then recursively copies the directories with the respective directories by adding *.bak* as the extensions to the directories.
- Output:

```
ls -p
test1/ test1.bak. test2/ test2.bak/ test.txt
```

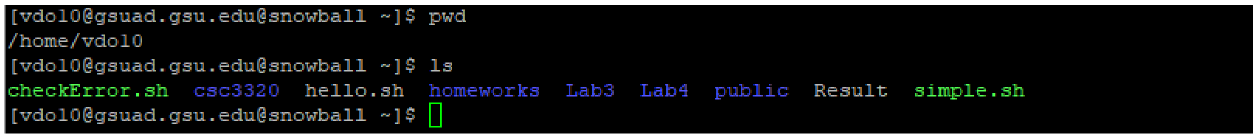
    - (-p puts a slash after every directory listed)

Part III Programming: 15pts

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

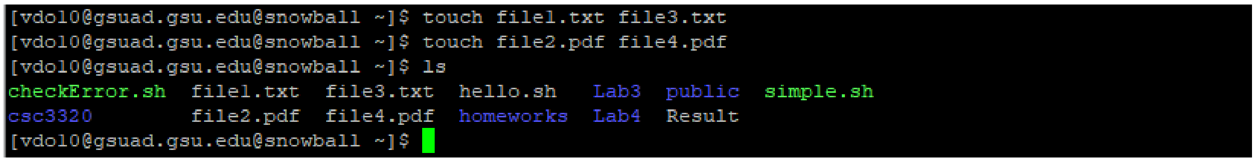
a. A copy of each file in that folder must be made. Append the string “_copy” to the name of the file
b. The duplicate (copied) files must be in separate directories with each directory specifying the type of the file (e.g. txt files in directory named txtfiles, pdf files in directory named pdffiles 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.



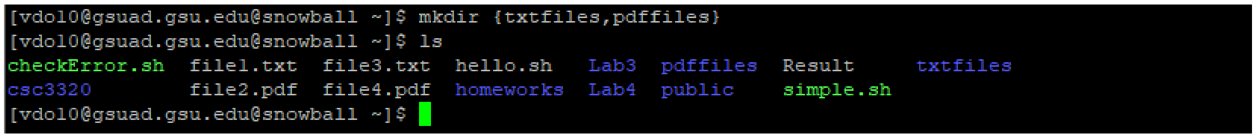
```
[vdo10@gsuad.gsu.edu@snowball ~]$ pwd
/home/vdo10
[vdo10@gsuad.gsu.edu@snowball ~]$ ls
checkError.sh  csc3320  hello.sh  homeworks  Lab3  Lab4  public  Result  simple.sh
[vdo10@gsuad.gsu.edu@snowball ~]$ 
```

I used the `pwd` and `ls` commands to check all of the directories in my current working directory before I started programming this homework.



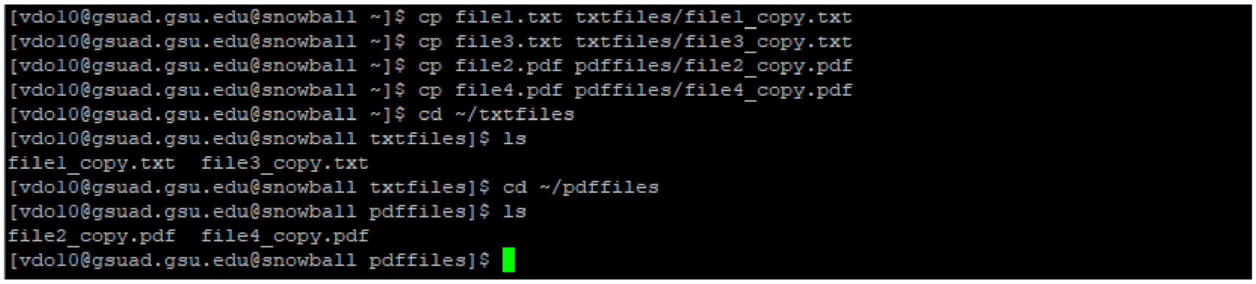
```
[vdo10@gsuad.gsu.edu@snowball ~]$ touch file1.txt file3.txt
[vdo10@gsuad.gsu.edu@snowball ~]$ touch file2.pdf file4.pdf
[vdo10@gsuad.gsu.edu@snowball ~]$ ls
checkError.sh  file1.txt  file3.txt  hello.sh   Lab3  public  simple.sh
csc3320        file2.pdf  file4.pdf  homeworks  Lab4  Result
[vdo10@gsuad.gsu.edu@snowball ~]$ 
```

I made up some random .txt and .pdf files using `touch` command and checked to see if they were actually there by using `ls`.



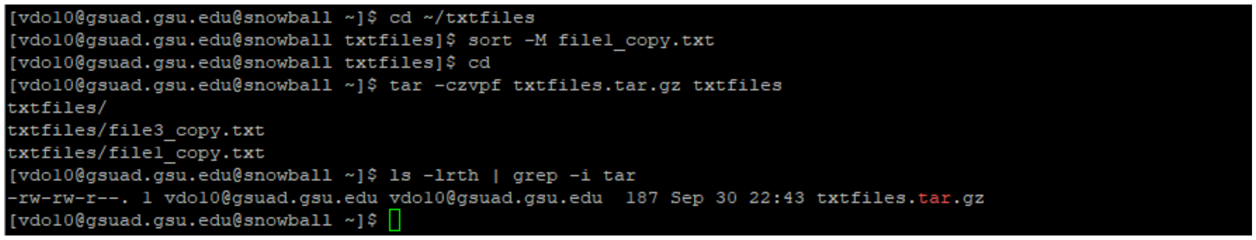
```
[vdo10@gsuad.gsu.edu@snowball ~]$ mkdir {txtfiles,pdffiles}
[vdo10@gsuad.gsu.edu@snowball ~]$ ls
checkError.sh  file1.txt  file3.txt  hello.sh   Lab3  pdffiles  Result     txtfiles
csc3320        file2.pdf  file4.pdf  homeworks  Lab4  public    simple.sh
[vdo10@gsuad.gsu.edu@snowball ~]$ 
```

I made two separate directories named *txtfiles* and *pdffiles* using the `mkdir` command and checked back at the home directory using `ls`.



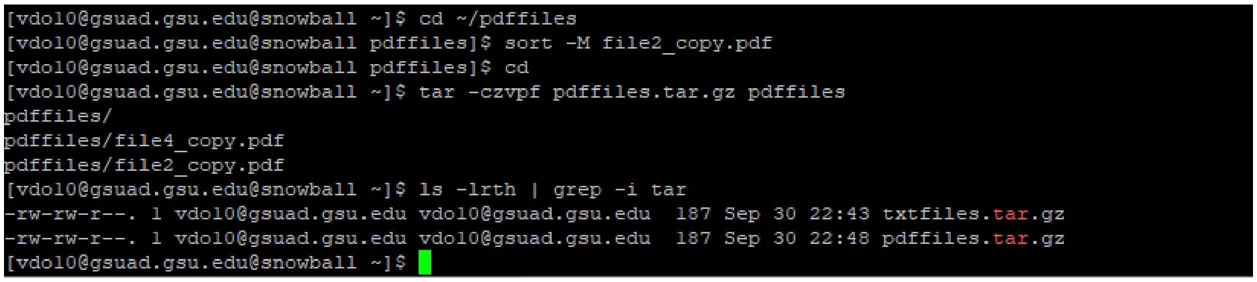
```
[vdol0@gsuad.gsu.edu@snowball ~]$ cp file1.txt txtfiles/file1_copy.txt
[vdol0@gsuad.gsu.edu@snowball ~]$ cp file3.txt txtfiles/file3_copy.txt
[vdol0@gsuad.gsu.edu@snowball ~]$ cp file2.pdf pdffiles/file2_copy.pdf
[vdol0@gsuad.gsu.edu@snowball ~]$ cp file4.pdf pdffiles/file4_copy.pdf
[vdol0@gsuad.gsu.edu@snowball ~]$ cd ~/txtfiles
[vdol0@gsuad.gsu.edu@snowball txtfiles]$ ls
file1_copy.txt  file3_copy.txt
[vdol0@gsuad.gsu.edu@snowball txtfiles]$ cd ~/pdffiles
[vdol0@gsuad.gsu.edu@snowball pdffiles]$ ls
file2_copy.pdf  file4_copy.pdf
[vdol0@gsuad.gsu.edu@snowball pdffiles]$
```

I copied the files to their respective directories using the `cp` command (i.e *file1.txt* would be copied into *txtfiles* as *file1_copy.txt*), went into the directories using the `cd ~/txtfiles` and `cd ~/pdffiles` commands, and checked for the duplicate copies using `ls`.



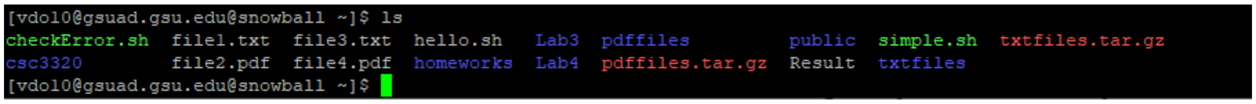
```
[vdol0@gsuad.gsu.edu@snowball ~]$ cd ~/txtfiles
[vdol0@gsuad.gsu.edu@snowball txtfiles]$ sort -M file1_copy.txt
[vdol0@gsuad.gsu.edu@snowball txtfiles]$ cd
[vdol0@gsuad.gsu.edu@snowball ~]$ tar -czvpf txtfiles.tar.gz txtfiles
txtfiles/
txtfiles/file3_copy.txt
txtfiles/file1_copy.txt
[vdol0@gsuad.gsu.edu@snowball ~]$ ls -lrth | grep -i tar
-rw-rw-r--. 1 vdol0@gsuad.gsu.edu vdol0@gsuad.gsu.edu  187 Sep 30 22:43 txtfiles.tar.gz
[vdol0@gsuad.gsu.edu@snowball ~]$
```

I sorted the *txtfiles* in chronological order of months by using the `sort -M` command and then archived this file of directory using the `tar` command.



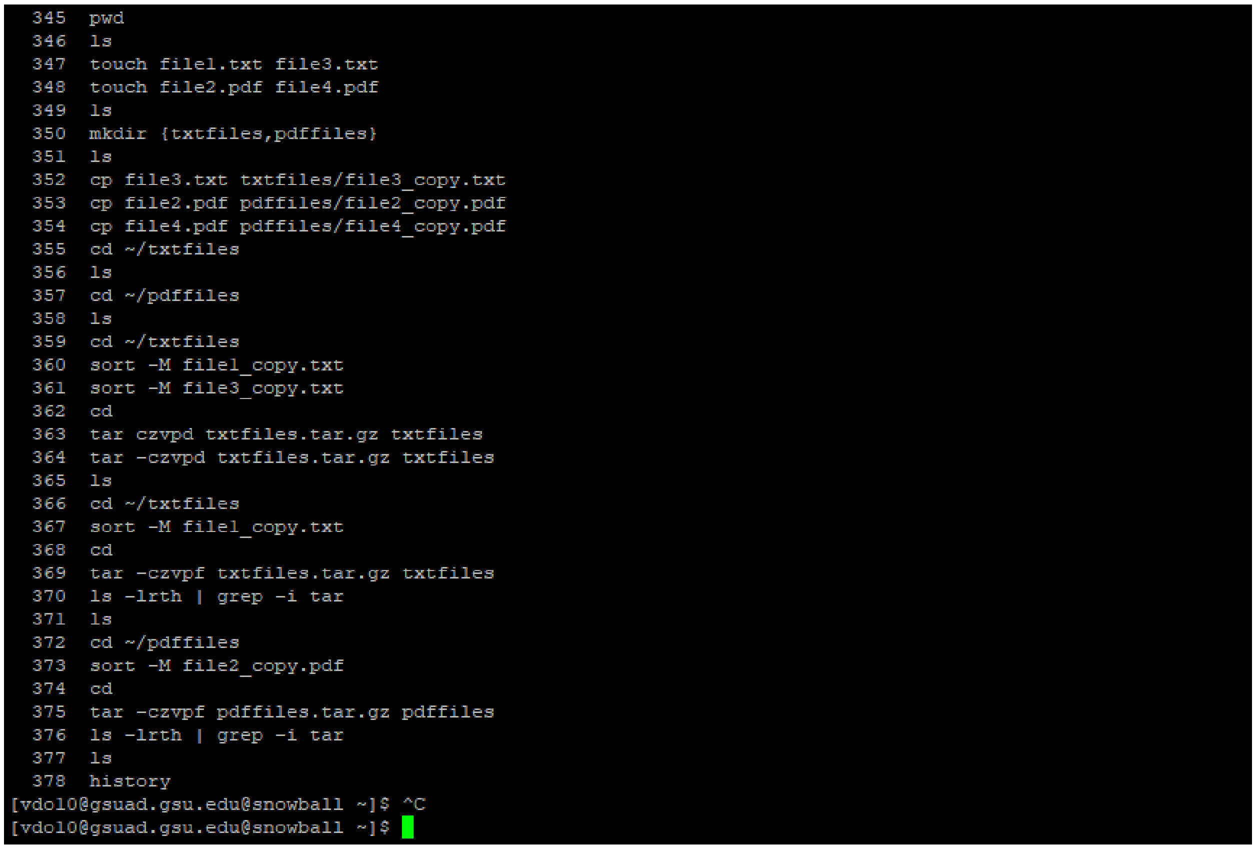
```
[vdol0@gsuad.gsu.edu@snowball ~]$ cd ~/pdffiles
[vdol0@gsuad.gsu.edu@snowball pdffiles]$ sort -M file2_copy.pdf
[vdol0@gsuad.gsu.edu@snowball pdffiles]$ cd
[vdol0@gsuad.gsu.edu@snowball ~]$ tar -czvpf pdffiles.tar.gz pdffiles
pdffiles/
pdffiles/file4_copy.pdf
pdffiles/file2_copy.pdf
[vdol0@gsuad.gsu.edu@snowball ~]$ ls -lrth | grep -i tar
-rw-rw-r--. 1 vdol0@gsuad.gsu.edu vdol0@gsuad.gsu.edu  187 Sep 30 22:43 txtfiles.tar.gz
-rw-rw-r--. 1 vdol0@gsuad.gsu.edu vdol0@gsuad.gsu.edu  187 Sep 30 22:48 pdffiles.tar.gz
[vdol0@gsuad.gsu.edu@snowball ~]$
```

I did the exact same thing as the previous screenshot step above, but with the *pdffiles* instead.



```
[vdol0@gsuad.gsu.edu@snowball ~]$ ls
checkError.sh  file1.txt  file3.txt  hello.sh   Lab3  pdffiles         public  simple.sh  txtfiles.tar.gz
csc3320        file2.pdf  file4.pdf  homeworks  Lab4  pdffiles.tar.gz  Result  txtfiles
[vdol0@gsuad.gsu.edu@snowball ~]$
```

I used `ls` to check for the *tar* files in the home directory.



```
  345  pwd
  346  ls
  347  touch file1.txt file3.txt
  348  touch file2.pdf file4.pdf
  349  ls
  350  mkdir {txtfiles,pdffiles}
  351  ls
  352  cp file3.txt txtfiles/file3_copy.txt
  353  cp file2.pdf pdffiles/file2_copy.pdf
  354  cp file4.pdf pdffiles/file4_copy.pdf
  355  cd ~/txtfiles
  356  ls
  357  cd ~/pdffiles
  358  ls
  359  cd ~/txtfiles
  360  sort -M file1_copy.txt
  361  sort -M file3_copy.txt
  362  cd
  363  tar czvpd txtfiles.tar.gz txtfiles
  364  tar -czvpd txtfiles.tar.gz txtfiles
  365  ls
  366  cd ~/txtfiles
  367  sort -M file1_copy.txt
  368  cd
  369  tar -czvpf txtfiles.tar.gz txtfiles
  370  ls -lrth | grep -i tar
  371  ls
  372  cd ~/pdffiles
  373  sort -M file2_copy.pdf
  374  cd
  375  tar -czvpf pdffiles.tar.gz pdffiles
  376  ls -lrth | grep -i tar
  377  ls
  378  history
[vdo10@gsuad.gsu.edu@snowball ~]$ ^C
[vdo10@gsuad.gsu.edu@snowball ~]$
```

Here is the history of my commands.