IC221 Lab1 Shell Commands Name: Umberto
AY24 Spring, 120 points total

Setup

Create a working folder and `cd` into it
Download `lab.tar.gz` to your working folder
Extract the files: `tar -xzvf lab.tar.gz`
`cd stu`

Submission

Submit this completed worksheet to `https://submit.usna.edu`

Part 1: The Manual Pages (30)

Unix/Linux systems are well documented through a set of *manual pages*. To access the manuals, use the man command. For example, to view the manual page for ls:

```
$ man ls
```

This brings up the manual page, whose header looks like this:

```
NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory by default).  Sort entries
alphabetically if none
       of -cftuvSUX nor --sort is specified.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..
```

The manual page provides both a brief description of the command and the arguments and options. For example, we see that the option `-a` or `--all` both list all entries for the directory, including `.` and `..` and all hidden files starting with `.` while, conversely, the `-A` or `--almost-all` option list all hidden files while *not* displaying the `.` or `..` entries.

This is just one of many options for the `ls` command, and you can scroll down using the up and down arrow key.

Quit the manual page by pressing `q`.

Lab Questions and Tasks

Perform the following tasks and answer the following questions:

1. (10 points) For the `ls` command, what option prints information out in long form, like `-l`, but does *not* print any group ownership information?

```
list -o like -l, but do not list group information
```

Provide a copy of the output using ls with this option run from the top level of the lab directory.

```
➜  stu git:(main) ✗ ls -l
total 4320
-r-xr-xr-x 1 amnesia amnesia       0 Jan  4  2024 Icon_
-rwxrwxrwx 1 amnesia amnesia      11 Jan  4  2024 beatarmy.txt
-rwxrwxrwx 1 amnesia amnesia  936251 Jan  4  2024 dickens.txt
-rwxrwxrwx 1 amnesia amnesia       9 Jan  4  2024 gonavy.txt
-rwxrwxrwx 1 amnesia amnesia    3243 Jan  4  2024 hemingway.txt
-rwxrwxrwx 1 amnesia amnesia   20425 Jan  4  2024 lab1.docx
-rwxrwxrwx 1 amnesia amnesia 3399671 Jan  4  2024 netflix_titles.csv
drwxrwxrwx 1 amnesia amnesia    4096 Jan  4  2024 part1
-rwxrwxrwx 1 amnesia amnesia   59476 Jan  4  2024 sample-db.csv
```

2. Change into the `part1` directory and type `ls`. You will see a list of files: `a b c d e f g`. Note that `ls` lists the files in alphabetic order.

(a) (4 points) What `ls` option will list the files in *reverse* alphabetic order?

```
ls -r, ls --reverse: reverse order while sorting
```

Provide a copy of your output of your `ls` with the *reverse* option, and the addition of `-l`.

```
➜  part1 git:(main) ✗ ls -l -r
total 108
-rwxrwxrwx 1 amnesia amnesia  9894 Jan  4  2024 g
-rwxrwxrwx 1 amnesia amnesia 13946 Jan  4  2024 f
-rwxrwxrwx 1 amnesia amnesia 16989 Jan  4  2024 e
-rwxrwxrwx 1 amnesia amnesia 22156 Jan  4  2024 d
-rwxrwxrwx 1 amnesia amnesia 12214 Jan  4  2024 c
-rwxrwxrwx 1 amnesia amnesia 12322 Jan  4  2024 b
-rwxrwxrwx 1 amnesia amnesia  6091 Jan  4  2024 a
-r-xr-xr-x 1 amnesia amnesia     0 Jan  4  2024 Icon_
```

(b) (3 points) What `ls` options will sort the files by size, from largest to smallest?

```
list -S, list --size: sort by file size, largest first
```

Provide a copy of your output of your `ls` by file size, with the addition of `-l`.

```
➜  part1 git:(main) ✗ ls -l -S
total 108
-rwxrwxrwx 1 amnesia amnesia 22156 Jan  4  2024 d
-rwxrwxrwx 1 amnesia amnesia 16989 Jan  4  2024 e
-rwxrwxrwx 1 amnesia amnesia 13946 Jan  4  2024 f
-rwxrwxrwx 1 amnesia amnesia 12322 Jan  4  2024 b
-rwxrwxrwx 1 amnesia amnesia 12214 Jan  4  2024 c
-rwxrwxrwx 1 amnesia amnesia  9894 Jan  4  2024 g
```

```
-rwxrwxrwx 1 amnesia amnesia  6091 Jan  4  2024 a
-r-xr-xr-x 1 amnesia amnesia     0 Jan  4  2024 Icon_
```

(c) (3 points) What ls option(s) will sort the files in *reverse size order* (from smallest to largest)?

```
ls -S -r
```

Provide a copy of your output of your `ls` in reverse size order, with the addition of `-l`.

```
➜  part1 git:(main) ✗ ls -l -S -r
total 108
-r-xr-xr-x 1 amnesia amnesia     0 Jan  4  2024 Icon_
-rwxrwxrwx 1 amnesia amnesia  6091 Jan  4  2024 a
-rwxrwxrwx 1 amnesia amnesia  9894 Jan  4  2024 g
-rwxrwxrwx 1 amnesia amnesia 12214 Jan  4  2024 c
-rwxrwxrwx 1 amnesia amnesia 12322 Jan  4  2024 b
-rwxrwxrwx 1 amnesia amnesia 13946 Jan  4  2024 f
-rwxrwxrwx 1 amnesia amnesia 16989 Jan  4  2024 e
-rwxrwxrwx 1 amnesia amnesia 22156 Jan  4  2024 d
```

3. (5 points) Remove the `g` file using the `rm` command. Notice that the shell asked you to confirm removing the item. Look at the manual page for `rm`. What option can you use to avoid having to confirm the removal of an item?

rm -f, rm –-force: ignore nonexistent files and arguments, never prompt

4. (5 points) Read the manual page for the `touch` command. One of the uses for `touch` is to update the last modified timestamp of a file (you can view that last modified using `ls -l`). Use the `touch` command to create a file `y2k` whose last modification time was Dec. 31 1999 at 23:59.59. Include the command you used on your worksheet and a copy of your `ls -l` output of the `y2k` file.

Command Used:

```
touch --date="1999-12-31 23:59:59" y2k
```

Output of `ls -l`:

```
➜  part1 git:(main) ✗ ls -l --full-time
total 96
-r-xr-xr-x 1 amnesia amnesia     0 2024-01-04 20:35:22.000000000 +0100 Icon_
-rwxrwxrwx 1 amnesia amnesia  6091 2024-01-04 20:35:22.000000000 +0100 a
-rwxrwxrwx 1 amnesia amnesia 12322 2024-01-04 20:35:22.000000000 +0100 b
-rwxrwxrwx 1 amnesia amnesia 12214 2024-01-04 20:35:22.000000000 +0100 c
-rwxrwxrwx 1 amnesia amnesia 22156 2024-01-04 20:35:22.000000000 +0100 d
-rwxrwxrwx 1 amnesia amnesia 16989 2024-01-04 20:35:22.000000000 +0100 e
-rwxrwxrwx 1 amnesia amnesia 13946 2024-01-04 20:35:22.000000000 +0100 f
-rwxrwxrwx 1 amnesia amnesia     0 1999-12-31 23:59:59.000000000 +0100 y2k
```

Part 2: Text File Operations (30)

Some Unix/Linux shell commands are used for viewing and changing the contents of files.

Viewing Files

The command to display a file's contents is the `cat` command, which is short for concatenate. Here is the man page synopsis for cat:

```
NAME
     cat -- concatenate and print files

SYNOPSIS
     cat [-benstuv] [file ...]

DESCRIPTION
     The cat utility reads files sequentially, writing them to the standard output.  The
file operands
     are processed in command-line order.  If file is a single dash (`-') or absent, cat
reads from the
     standard input.  If file is a UNIX domain socket, cat connects to it and then reads it
until EOF.
     This complements the UNIX domain binding capability available in inetd(8).
```

The `cat` command takes a file or sequence of files and writes them to *standard out*, which is the terminal. Let's do a quick example. Navigate to part2 in the lab directory, and let's `cat` the output of the `gonavy.txt` file:

```
$ cat gonavy.txt
Go Navy!
```

`cat` can also take multiple files as input, and print their contents to the terminal one after the other, or *concatenate* the contents by printing it all to standard output. Let's use the `cat` command to view the contents of the files in `part2` of the lab directory. Use `cd` to navigate there.

```
$ cat gonavy.txt beatarmy.txt
Go Navy!
Beat Army!
```

The contents of `beatarmy.txt` is "Beat Army!" and the contents of `gonavy.txt` is "Go Navy!". The concatenation of those contents is "Go Navy! Beat Army!" across two lines.

Viewing files with `less` and `more`

One drawback of viewing files with cat is that it clutters up the terminal output. You could always clear the terminal using `clear` or `Control-l` (go ahead and try that now), but it can get bothersome the more you need to do so.

Unix provides two ways to view a file *iteratively* (not all at once) within a terminal application: `less` and `more`. The basic difference between the two file viewers is that `less` allows you to go forward and backwards in a file while `more` only allows you to move forward in the file, exiting at the end. Thus, in Unix/Linux, `less` is really more.

Let's see an example of why this is useful. Consider two great authors of literature, Charles Dickens and Ernest Hemingway. Dickens was paid by the word and so his stories are very long indeed, while Hemingway was a minimalist, and his stories were quite short. In the `part2` directory you have two text files, one named `dickens.txt` and one named `hemingway.txt`.

We can easily read `hemingway.txt` using `more`, moving forward in the file by pressing `<space>` to page down, or by using the down-arrow key. The indicator at the bottom of the screen describes how far in the file we've progressed.

Let's now use `more` to read `dickens.txt` … oh man! This is going to take forever, and there is only one way to go, forward. Clearly, we need a more powerful viewer, so we use `less`. The `less` command allows you to move forward and back within the file, plus a bunch of other useful navigation tools. Here are some:

- Quit: **`q`**
- Search forward: **`/`** then type your search (regex allowed) then use the following
    - Next match going forwards: **`n`**
    - Next match going backwards: **`N`**
- Search backward: **`?`** then type your search (regex allowed) then use the following
    - Next going backwards: **`n`**
    - Next match going forwards: **`N`**
- Go to line: **`:`** then type line number
- Start of file: **`<`**
- End of file: **`>`**
- Panic: **`CTRL-g`**

Lab Questions and Tasks:

1. (5 points) Use `cat` to output a "Beat Army!", then Hemingway's story, then "Go Navy!" at the end. Use only a single command. Show the command you used:

```
cat beatarmy.txt hemingway.txt gonavy.txt
```

2. (5 points) Why is `less` more?

Because while more only allows you to move forward in a file, exiting at the end, less allows you to move forward and backwards, and allows a bunch of useful navigation tools (search for matches forward and backwards, go to line, go to start and end of file, etc.).

3. Use `less` to open `dickens.txt`.

(a) (5 points) Search for the first instance of "Fagin". What is the full text of the sentence in which it first occurs?

'Greenland. Is Fagin upstairs?'

(b) (5 points) Find the second to last instance of "Fagin". Describe how you did that:

I moved to the end of the file with “>”, then I used the search backwards utility by typing ?Fagin, then I typed “n” to move to the second to last instance.

Copy the text of the sentence it appears in:

As far from home, died the chief remaining members of his friend Fagin's gang.

(c) (10 points) Go to line 2691. What is the name of that chapter?

CHAPTER IX

CONTAINING FURTHER PARTICULARS CONCERNING THE PLEASANT OLD GENTLEMAN, AND HIS HOPEFUL PUPILS

Part 3: Viewing Portions of Files (30)

When we do want to print the contents of a file to the terminal, we may not want to print the whole thing, as `cat` does. Instead, sometimes we'd like just to print the first *n* lines, or the last *n* lines, or some set of lines in the middle, or just print lines that match a given search string. For that we have a set of very useful commands.

For the following examples, navigate to the `part3` directory in the lab directory. There is a sample file `sample-db.csv` that you will use for this part that contains fake records of people entering information on a web server.

View the first or last *n* lines with `head` or `tail`

The `head` command is used to print the 'head' of the file. By default, `head` prints the first 10 lines:

```
$ head sample-db.csv
```

Similarly, `tail` by default will show the last 10 lines:

```
$ tail sample-db.csv
```

You can describe how many lines you wish to show in two ways, either by using `-n` argument, where n is replaced by the number of lines. For example, to print the first 3 lines:

```
$ head -3 sample-db.csv
```

Or, by passing the number of lines after `-n`

```
$ head -n 3 sample-db.csv
```

Printing intermediate lines with sed

The `sed` command is very powerful, and it has many more features than just printing intermediary lines. Here is the format of the `sed` command:

```
Line Number Input
      |          ,--File to process
      v          v
 sed -n 3,10p filename
        ^  ^^
Start---'  ||
Finish-----''---Print those lines
```

As an example, what if we want to print lines 2 through 4 of the file only:

```
$ sed -n 2,4p sample-db.csv
Donette,Foller,Printing Dimensions,34 Center St,Hamilton,Butler,OH,45011,513-570-1893,513-549-4561,donette.foller@cox.net,http://www.printingdimensions.com
Mitsue,Tollner,Morlong Associates,7 Eads St,Chicago,Cook,IL,60632,773-573-6914,773-924-8565,mitsue_tollner@yahoo.com,http://www.morlongassociates.com
Leota,Dilliard,Commercial Press,7 W Jackson Blvd,San Jose,Santa Clara,CA,95111,408-752-3500,408-813-1105,leota@hotmail.com,http://www.commercialpress.com
```

Printing only matching lines with `grep`

Finally, we need a mechanism to only process lines that match a condition. The `grep` command is used for that.

For example, let's consider trying to just print the lines where the person is from the state of New Jersey. To do that, we need to first identify a unique part of lines for people from New Jersey, and that is "NJ" in the address field.

```
$ grep NJ sample-db.csv
Art,Venere,Chemel, James L Cpa,8 W Cerritos Ave #54,Bridgeport,Gloucester,NJ,08014,856-636-8749,856-264-4130,art@venere.org,http://www.chemeljameslcpa.com
Alisha,Slusarski,Wtlz Power 107 Fm,3273 State St,Middlesex,Middlesex,NJ,08846,732-658-3154,732-635-3453,alisha@slusarski.com,http://www.wtlzpowerfm.com
Ernie,Stenseth,Knwz Newsradio,45 E Liberty St,Ridgefield Park,Bergen,NJ,07660,201-709-6245,201-387-9093,ernie_stenseth@aol.com,http://www.knwznewsradio.com
(...)
```

Note that in a `grep` command, the first part is the search term and the second part is the file to be searched. The `grep` command can also be used with a special search language called *regular expressions*, which allow you to search for all sorts of things.

Lab Questions and Tasks

1. (10 points) Using the man pages for `head` and `tail`, produce a command line to print the first kilobyte of the file `netflix_titles.csv`. A kilobyte is 2^{10} or 1024 bytes.
Command used:

```
head -c 1024 netflix_titles.csv
```

2. (10 points) Use `less` or `grep` to find the line number of the entry in `netflix_titles.csv` containing the text "V for Vendetta". Be sure to enclose your search term in quotes.

Line Number:

```
8672
```

Command(s) Used:

```
grep -n "V for Vendetta" netflix_titles.csv
```

3. (10 points) Produce a `sed` command to just print the line with "V for Vendetta" and the 2 lines that follow it. Note that the movie listing numbers will not exactly match the text line numbers.
Command:

```
sed -n 8672,8674p netflix_titles.csv
```

Part 4: Pipelines and Output Processing (30)

An important tool for file processing is to be able to take the output of processing one file and set the result as the input to another process. These process parts can be chained together into a pipeline. Consider this simple pipeline below:

```
$ cat sample-db.csv | head -20
```

The pipe or `|` takes the output of one command and sets it as the input of another. In the example above, the output of the `cat` is to print the whole contents of the file to the input of `head`, which then only prints the first 20 lines of its input. While this is a contrived example, you should start to see the power of the pipeline. Consider the below command:

```
$ grep NJ sample-db.csv | wc -l
```

The first part of the command will print out only the lines that contain the pattern "NJ", this output is then set as the input to the `wc` command, which is a command line tools to count words, lines, and bytes. The `-l` option says to just print the line count, and thus, the command above prints out the number of lines.

Nearly all Unix command line tools have an option to either read from a file or from standard input along a pipeline. For example, the above command can be rewritten with `cat` at the front, as follows:

```
$ cat sample-db.csv | grep NJ | wc -l
```

Parsing just certain fields with `cut`

A very useful pipeline tool is `cut`, which is used to extract fields from a formatted file, like our database file. Here is a basic command line argument:

```
          ,--- delimiter
        __|_            ,-----Input File, or leave off to read from stdin
       /    \           v
$ cut -d "," -f 1 sample-db.csv | head -5
             \__/
               \__.-- Field
```

The delimiter determines how the file is to be cut. The `sample-db.csv` file is a comma separated file, so it is delimited by commas; that is, every item in the line is separated by comma to distinguish it from other items on the line. The above command will print the first 5 lines of output from the first delimited item:

```
$ cut -d "," -f 1 sample-db.csv | head -5
Lenna
Donette
Mitsue
Leota
Sage
```

Sorting and Removing Duplicates

Two parsing tools we'll use for sorting in this lab are `sort` and `uniq`. The former will sort input and the latter will remove any *adjacent* duplicate lines. Check out their man pages to see how they might be used in conjunction, to solve different file parsing problems.

Transposing text with `tr`

The `tr` command is the *translate character* tool. It takes two arguments:

```
tr <from> <to>
```

It will read from standard input and convert any instances of “from” to “to”. For example, if we wish to list phone numbers we find:

```
$ cut -d, -f 9 sample-db.csv | head
907-385-4412
513-570-1893
773-573-6914
408-752-3500
605-414-2147
631-335-3414
310-498-5651
440-780-8425
602-277-4385
```

And if we wish to replace the hyphen with a space, we can do so easily like this:

```
$ cut -d, -f 9 sample-db.csv | tr "-" " " | head
907 385 4412
513 570 1893
773 573 6914
408 752 3500
605 414 2147
631 335 3414
310 498 5651
440 780 8425
602 277 4385
```

Lab Questions and Tasks

1. (10 points) Create a pipeline to count the number of unique states represented in the database file. Use `sort -u` to identify unique states.

Answer:

```
45
```

Command Used:

```
cut -d "," -f 7 sample-db.csv | sort -u | wc -w
```

2. (10 points) How many first names (first field) in the file repeat?

Answer:

```
10
```

Command Used:

```
cut -d "," -f 1 sample-db.csv | sort | uniq -d | wc -l
```

3. (10 points) Write a pipeline to print to the terminal a reverse-sorted list (larger numbers followed by smaller numbers) of all the unique telephone area codes found in field 9.

Command Used:

```
cut -d "," -f 9 sample-db.csv | sort -r | uniq
```