**03 – Filters, Scripts and Pipes**

**Activities**

COMP190 – Tools and Techniques for Software Development

Dickinson College

Fall 2022

**Name:**

In the article *Linux vs. Unix: What’s the Difference*, Phil Estes summarized the Unix philosophy as “utilizing small, purpose-built programs in combination to do complex overall tasks.” In the past homework assignments you’ve learned about a few of these “purpose-built” programs (i.e. *filters*). In this assignment you will gain experience with *redirection*, *shell scripts* and *pipes* that allow you to combine filters to complete more complex tasks. You’ll also learn about a few more filters and use them to solve some challenges. Along the way you’ll gain practice with Linux/Unix file permissions and with how to run and terminate background processes.

**Getting Setup:**

The activities in this homework will require that you have a new file to work with. Use an editor (e.g. nano, Mousepad) to create a file named <name>-music.txt in your home directory.

In that file, each line should contain a number followed by a period, the title of one of your favorite songs, then a comma and then the name of the band or artist that performs it. For example, here are a few lines from a file that I created:

1. The Thrill is Gone, BB King

2. Me and the Devil Blues, Robert Johnson

3. I'd Rather Go Blind, Etta James

4. When Love Comes to Town, BB King

Your file must:

* Contain at least 10 songs
* Have at least 3 lines with different song titles but the same band/artist
* If a title or band or artist name contains a comma, it should be omitted.

If you are not a big music fan and have trouble creating this file, feel free to do a web search for songs or just make up some song titles and artists.

1. Use the cat command to display your <name>-music.txt file in a Terminal. Paste a screenshot showing your cat command and its output below.

**Redirection:**

The *redirection* operators in Linux/Unix can be used to allow a command to accept its input from a file rather than from standard input (<) or to have the output from a command be written into a file (>) or appended to a file (>>) instead of appearing on standard output (i.e. in the terminal).

2. Consider the commands shown in the table below. Fill in the Input and Output columns with either the name of the file that will be used, or standard input or standard output as appropriate.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | **Command** | **Input** | **Output** |  |
|  | whoami > me.txt |  |  |  |
|  | sort < words.txt |  |  |  |
|  | head -n4 < lines.txt > top.txt |  |  |  |
|  |  |  |  |  |

3. Use the ls command with redirection to place a list of the files and directories in your home directory into a file named files.txt. Then display the contents of the files.txt file in the terminal and confirm that your command worked correctly. Paste a screenshot of the commands you used and their output below.

4. Use the man command or your favorite search engine to learn about the tail command. Then use redirection to display the names of the last five songs in your <name>-music.txt file from #3 in the terminal window. Paste a screenshot of the command you used and its output below.

5. Use the head and tail commands and redirection to place lines 1 and 2 and 9 and 10 from your <name>-music.txt file into a file named ends.txt. Then display the contents of the ends.txt file in the terminal and confirm that your commands worked correctly. Paste a screenshot of the commands you used and their output below.

6. Optional Extra: Make a copy of your <name>-music.txt file named copy.txt without using the cp command. Then display the contents of the copy.txt file in the terminal and confirm that your command worked correctly. Paste a screenshot of the commands you used and their output below.

**Shell Scripts:**

A shell script is a text file that contains a list of commands to be executed by a shell (i.e. command interpreter). Shell scripts provide a convenient way to package a collection of commands that perform a useful task, so that you don’t have to type them all each time. More sophisticated shell scripts can even contain programming constructs like loops, conditionals and reusable functions.

In completing the questions in the next several sections, you will build a shell script that would help with a music countdown show. These shows countdown of the top so many songs (e.g. 10, 40 or 100) of the week (or year, or decade, etc.). The show plays each of the songs in reverse order of preference (i.e. they count down from 10 to 1).[[1]](#footnote-1)

You'll build this script up piece by piece through the exercises. This will give you experience with how to create and test a script incrementally. It also provides an opportunity to learn about a few new topics and commands along the way. Ultimately, the script you build will display a title for the countdown with the current date and then a list of a specified number of your favorite songs, from your <name>-music.txt file, in reverse order.

7. Let’s get started.

* Create a new directory named scripts in your home directory.
* Move your <name>-music.txt file into the scripts directory.
* Make the scripts directory the working directory.
* List the files in the scripts directory to confirm that your <name>-music.txt file is there.

Take a screenshot of the commands you have used and their output and paste it below.

8. Now we’ll start creating the countdown script:

* Create a new text file named countdown.sh in the scripts directory.
* Add a line at the top of the file with a sha-bang and have it start the bash shell as is shown in the class slides.
* Add a command to the script that displays the first 5 lines the <name>-music.txt file in the terminal.

Take a screenshot of the editor showing your script thus far and paste it below.

9. Try to use the command below to run the script.

./countdown.sh

Take a screenshot of your command and the error message that is displayed and paste it below.

**File Permissions:**

In Linux/Unix every file has a set of *file permissions* that control which users have permission to perform which operations on that file. As we saw in the class slides, these permissions control whether the user, the group or others not in the group can read, write or execute the file.

10. The ls command with a -l flag (i.e. ls -l) will display a detailed listing of the files and directories in the current directory. This detailed listing includes information about the file’s owner, group and permissions. Consider the output of an ls -l command for the permex.txt file shown below:



Use the information in the above example to complete the table below.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Detail** | **Value** |  |
|  | Owner’s name |  |  |
|  | Group’s name |  |  |
|  | Owner’s permissions |  |  |
|  | Group’s permissions |  |  |
|  | Other’s permissions |  |  |
|  |  |  |  |

11. Complete the table below by giving a chmod command to set the permissions of the permex.txt file to the permissions indicated. For each row of the table, assume that the permissions on the permex.txt file begin as shown above in question #10. Try to use as short of a chmod command as possible (i.e. only make the necessary changes, do not just set all of the permissions). You can test your answers by creating a file of your own and setting its permissions.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Desired Permissions** | **chmod command** |  |
|  | rwxrwxr-- |  |  |
|  | rwxr-xr-x |  |  |
|  | rw-r-xr-- |  |  |
|  | r--r--r-- |  |  |
|  | rw-rw-rw- |  |  |
|  |  |  |  |

12. Use the ls -l command to show the details of the countdown.sh script. Take a screenshot of the output showing all of the details for the countdown.sh script below.

13. Using the information in your answer to #10, briefly explain why you received the error message that you did in #9 when you attempted to run the script.

14. Use the chmod command to give yourself (i.e. the owner) execute permission for the countdown.sh script. Use the ls -l command to confirm that you have done so successfully. Take a screenshot of the commands you have used and their output and paste it below.

15. Now run the script using the command from question #9. This time it should work. Take a screenshot of the commands you have used and their output and paste it below.

**Reversing the Order:**

The output from your script in question #15 should properly show the top 5 songs in your list. However, for our countdown script these songs need to be in reverse order.

16. The sort command filters its input by reordering its lines into sorted order and outputting them - I know, shocking! Use the manual pages to learn about the sort command. Use the sort command to display in the terminal the lines in your <name>-music.txt file in reverse order. Hint: Find the flag you need to use with sort. Take a screenshot of the command you used and its output and paste it below.

17. Now modify your countdown.sh script by redirecting the first 5 lines of <name>-music.txt into a file. This will be an *intermediate file* that you just use to hold the lines so that you can redirect them into the sort command so that they are displayed in the terminal in reverse order. You can name the intermediate file whatever you like, but its best if it is named something that is indicative of its purpose.

a. Give a screenshot of the code for your updated script below.

b. Run your script. It should now output the first 5 lines of your <name>-music.txt file in reverse order (5, 4, 3, 2, 1). Take a screenshot of the command you used and its output and paste it below.

18. Notice that now when you run your script the intermediate file that you created is left in your scripts directory. It would be better if the script deleted this file once it is no longer needed. Add a line to your script that deletes the intermediate file you created. Give a screenshot of the code for your updated script below.

**Pipes:**

Using redirection with intermediate files, as you did above, is sufficient for many scripting tasks. However, as tasks get more complicated the number of intermediate files needed grows. In many cases, these intermediate files serve no other purpose than directing the output of one filter to the input of another. So the overhead of naming, keeping track of and deleting these files can be burdensome.

*Pipes* are ideal for use in these situations because they allow the output of one filter to be routed directly to the input of another, without the creation of an intermediate file. For example, the command

tail --lines=5 < somefile.txt | sort

will output to the terminal the last five lines of the file somefile.txt in sorted order, without creating any

19. Use the head and tail commands with a pipe to display lines 4, 5, 6, and 7 from your <name>-music.txt file in the terminal window. Confirm that your command worked correctly and then paste a screenshot of the command you used and their output here.

20. Refactor your coutdown.sh script so that it accomplishes its task using a pipe and does not create an intermediate file. Your modified script should generate the same output as it did before your changes. Add a line to your script that deletes the intermediate file you created. Give a screenshot of the code for your updated script below.

**More Filters:**

There are lots of filters on a typical Linux/Unix system. You have seen a few of them at this point (cat, head, tail) We won’t cover all of the filters, but this section will gain practice learning about filters and using them to complete small task. Later exercises will have you combine them to do more complex tasks.

For each of the following filters, use the manual pages and/or your favorite search engine to learn about the command. Use what you find to answer the questions and complete the task.

wc

21. Use the manual pages and/or your favorite search engine to learn about the wc command. What does the wc command do?

22. Use the wc command to display in the terminal the number of lines that appear in your <name>-music.txt file. Use the appropriate flag(s) so that the output of your command is a single number indicating the number of lines. Check that the output agrees with the contents of your file. Take a screenshot of the command you used and its output and paste it below.

cut

23. Use the page linked below to learn about the cut command.

* <https://shapeshed.com/unix-cut/#how-to-cut-based-on-a-delimiter>

What does the cut command do?

24. Study the examples in the “How to cut based on a delimiter” section of the above page.

a. The delimiter is the character that separates the fields on a line. What flag (i.e. option) is used to set the delimiter for the cut command?

b. What flag is used to specify which fields should be cut (i.e. included in the output) by the cut command?

c. Use the cut command to display in the terminal a list of just the band or artists that appear in your <name>-music.txt file. The output should just be a list of the names that appear after the ‘,’ on each line in your file. Names that appear multiple times in your file should also appear multiple times in this output. Take a screenshot of the command you used and its output and paste it below.

25. Extend your command from #24c using a pipe and another filter so that the artists or bands displayed in alphabetical order. Note that this output should contain duplicates. Take a screenshot of the command you used and its output and paste it below.

uniq

26. Use the page linked below to learn about the uniq command.

* <https://www.redhat.com/sysadmin/uniq-command-lists>

What does the uniq command do?

27. Study the “Without any option” example in the above page. Extend your command from question #25, so that the output does not contain any duplicates. Take a screenshot of the command you used and its output and paste it below.

28. Write a command using filters and pipes that will show the number of different artists or bands that you have in your <name>-music.txt file. Take a screenshot of the command you used and its output and paste it below.

**Running and Terminating Programs**

All of the programs that we have run thus far were *short-running* programs - commands like ls, mkdir, etc. or filters like cat, sort, etc. When you typed their name at the command prompt, the program ran, did its thing and then terminated. When the program exited the command prompt returned and you could type another command. Some programs are *long-running*, these programs will continue running and will only terminate when you terminate them. This can happen when you close their window or force them to quit in a few other ways that you’ll see in this section. The techniques you learn for forcing programs to quit can also be very useful in terminating any program that has become non-responsive.

29. Any program that runs in a window will be a long running program. The Firefox web browser is one example. Launch Firefox from the terminal with the command:

firefox-esr

Note: If you are curious, esr stands for *extended service release* and just indicates that the version of Firefox that is running is one that will be supported for a longer period of time.

a. Find the terminal window behind the Firefox window. Ignore the two “Crash Annotation…” lines that have appear in the terminal - they are complaining about a missing library file. Did the command prompt return when Firefox was run? What happens if you press enter/return?

b. Terminate Firefox by closing its window. Does the command prompt return now?

30. When a program is running in the terminal and the command prompt has not returned you can often use the Ctrl-C (hold the Ctrl/Control key and press C) sequence to terminate the program. Run Firefox again. Make the terminal window active by clicking in it. Note that the command prompt has not returned and then press Ctrl-C. What happens?

31. Sometimes programs are frozen and are particularly stubborn about exiting. In those situations, you can force terminate them from another terminal window. This question will walk you through that process. Run the firefox command again, and note that the command prompt has not returned in the terminal. Now, open a second terminal window.

a. In this second terminal window, enter the command ps -u. The ps command shows a list of the *processes* (i.e. *running programs*). The -u flag tells ps to show all of the processes for your user. Take a screenshot of the ps command you used and its output and paste it below.

b. Every process in a Linux/Unix system as a *Process ID* (*PID*). What is the PID of the firefox-esr process?

c. When you know the PID of a process you can use the command:

kill -KILL <pid>

command to terminate the process. Use this command with the PID you found in part b. What happens?

**Summary:**

32. Complete the table below by filling in the Linux command or symbol that corresponds to each task.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Command** | **Task** |  |
|  |  | Redirect standard input from a file |  |
|  |  | Redirect standard output to a file |  |
|  |  | Redirect standard output and append to a file |  |
|  |  | Output the lines of a file in sorted order |  |
|  |  | Count the number of words or lines in a file |  |
|  |  | Select delimited fields from a file |  |
|  |  | Skip adjacent duplicate lines in a file |  |
|  |  | Display your running processes |  |
|  |  | Key combination to terminate a running process |  |
|  |  | Command to force terminate a running process |  |
|  |  |  |  |

**Optional:** To help us improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.

**Acknowledgements:**

Some materials, questions and resources have been adapted from activities posted on foss2serve.org.

* <http://foss2serve.org/index.php/Intro_to_Bash_(Activity)>
* <http://foss2serve.org/index.php/Linux_Beginner_Activity>

1. <https://www.youtube.com/watch?v=m7KXJATKdZQ> [↑](#footnote-ref-1)