CSCI-130 Linux Fundamentals - Week 4 (September 16, 2025) Assignment Submission

Assignment Details

1. Using a touch command, create several files in your user space in different directories.
2. Using a find command from different places, find the files you created on the first step.
3. Copy some of the files to the /tmp directory and run a find command starting from the root of directory tree to find those files. What do you see? Is this expected?
4. Repeat previous step and redirect result to a file. Use cat or less commands to see the content of the file.
5. Repeat this and redirect both, the stdout and stderr streams to the same file. Use cat or less commands to see the content of the file.
6. Describe your experience with the streams redirection.

How to submit

Submit relevant history entries with the appropriate comments when necessary.

Assignment Submission Notes

Steps 1 Through 5

History exported from terminal:

```
     1  mkdir -p ~/Documents/project1
    2  mkdir -p ~/Documents/project2
    3  mkdir -p ~/Downloads/temp
    4  mkdir -p ~/test_files
    5  touch file1.txt
    6  touch file2.txt
    7  touch ~/Documents/project1/data.txt
    8  touch ~/Documents/project1/config.txt
    9  touch ~/Documents/project2/notes.txt
   10  touch ~/Downloads/temp/download.txt
   11  touch ~/test_files/test1.txt
   12  touch ~/test_files/test2.txt
   13  echo 'Created directories and added files to them'
   14  find ~ -name '*.txt'
   15  find ~/Documents -name '*.txt'
   16  find . -name "test*.txt"
   17  find ~ -name "data.txt"
   18  find ~ -name "config.txt"
   19  echo 'Used find command from different locations to locate created
files'
   20  cp ~/file1.txt /tmp/
   21  cp ~/Documents/project1/data.txt /tmp/
   22  cp ~/test_files/test1.txt /tmp/
```

```
   23  find / -name "file1.txt"
   24  find / -name "data.txt"
   25  find / -name "test1.txt"
   26  echo 'Copied files to /tmp and searched from root - expected to see
permission denied errors'
   27  find / -name "*.txt" > all_txt_files.txt
   28  find / -name "file1.txt" > find_results.txt
   29  cat find_results.txt
   30  less find_results.txt
   31  find / -name "*.txt" > all_txt_files.txt
   32  less all_txt_files.txt
   33  echo 'Redirected find output to files and viewed with cat and less
commands'
   34  find / -name "file1.txt" > combined_output.txt 2>&1
   35  cat combined_output.txt
   36  find / -name "*.txt" > all_output.txt 2>&1
   37  less all_output.txt
   38  find / -name "data.txt" > only_stdout.txt
   39  find / -name "data.txt" > both_streams.txt 2>&1
   40  cat both_streams.txt
   41  echo 'Redirected both stdout and stderr to view the difference in
output handling'
   42  history > week_04_assignment_history.txt
```

Week 4 assignment history saved to `week_04_assignment_history.txt`.

Description of Stream Redirection Experience

When redirecting the output of the `find` command, I observed different behaviors based on how I handled the standard output (stdout) and standard error (stderr) streams. For example, when I redirected only stdout to a file, I captured the successful results of the command, but any error messages (such as permission denied errors) were not included. Conversely, when I redirected both stdout and stderr to the same file, I was able to see both the successful output and any error messages together, which provided a more comprehensive view of the command's execution. Using `2>&1` was particularly useful in this context, as it allowed me to consolidate all output into a single file for easier review. Additionally, I learned that using `>` replaces the content of the file, while `>>` would append to it, which is important to consider when managing output files. Overall, this exercise helped me understand the importance of stream redirection in managing command outputs effectively in a Linux environment.

Here are some examples of the commands used:

```
$ find / -name "file1.txt" > find_results.txt
# This command redirects only the standard output to find_results.txt
$ find / -name "file1.txt" > combined_output.txt 2>&1
# This command redirects both standard output and standard error to
combined_output.txt
```

I anticipate using redirection in future tasks to capture logs and error messages for troubleshooting and analysis much more effectively. For example, when I'm trying to run Ruby or Python scripts that seem to be

failing, I can redirect both output and errors to a log file to diagnose the issue without cluttering the terminal.

Here is an example for Ruby:

```
$ ruby my_script.rb > script_output.log 2>&1
# This captures both the output and any errors from the Ruby script into
script_output.log
```

Here is an example for Python:

```
$ python my_script.py > script_output.log 2> errors.log
# This captures standard output in script_output.log and errors in
errors.log files
```