CSCI-130 Linux Fundamentals - Week 5 (September 23, 2025) Assignment Submission

Assignment Details

1. Using examples we did in the class, practice a xargs command to search the content of the files (within the files) obtained by running the find command. Use the unnamed pipes to do so.
2. Describe the difference between using xargs and not using xargs command.
3. Use the following command to download a sample of the apache HTTP log file (wget):
4. wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Aug95.gz
5. After you downloaded the NASA_access_log … file, run the following to decompress it:
6. gunzip -d NASA_access_log_Aug95.gz (or gzip –d…)
7. Open the file and make yourself familiar with its format. Note that the first field in the file is the IP address of the requestor.
8. Keep that file for future use.

How to submit

Submit relevant history entries with the appropriate comments when necessary.

Assignment Submission Notes

Difference between using xargs and not using xargs command

Using `xargs` allows you to build and execute command lines from standard input. When you use `xargs`, it takes the output of one command (like `find`) and passes it as arguments to another command (like `grep`). This is particularly useful when dealing with a large number of files, as it can handle more arguments than a single command line can.

On the other hand, not using `xargs` means that you might be limited by the maximum command line length. If the output of the first command is too large, it could exceed this limit, causing the command to fail. Additionally, without `xargs`, you may need to use a loop to process each file individually, which can be less efficient.

Example of using xargs with find and grep

```
find . -name "*.txt" | xargs grep "search_term"
```

In this example, `find` locates all `.txt` files in the current directory and its subdirectories, and `xargs` passes those file names to `grep` to search for "search_term" within those files.

Example of not using xargs with find and grep

```
```

```
for file in $(find . -name "*.txt"); do
    grep "search_term" "$file"
done
```

In this example, we use a loop to iterate over each `.txt` file found by `find` and run `grep` on each file individually. This can be less efficient and may run into issues if there are too many files.

Downloading and decompressing the NASA access log file

```
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Aug95.gz
gunzip -d NASA_access_log_Aug95.gz
```

This will download the NASA access log file and decompress it, making it ready for review.

The first field in each line of the log file represents the IP address of the requestor, which can be useful for various analyses, such as tracking access patterns or identifying potential security issues. The second field typically contains the timestamp of the request. The third field usually indicates the request method (e.g., GET, POST) and the requested resource. The fourth field often contains the HTTP status code returned by the server. The fifth field may include the size of the response in bytes.

For example, a line in the log file might look like this:

```
in24.inetnebr.com - - [01/Aug/1995:00:00:01 -0400] "GET
/shuttle/missions/sts-68/news/sts-68-mcc-05.txt HTTP/1.0" 200 1839
```

In this example:

- `in24.inetnebr.com` is the IP address or hostname of the requestor.
- `- -` are placeholders. I'm not sure what they represent in this context.
- `[01/Aug/1995:00:00:01 -0400]` is the timestamp of the request.
- `"GET /shuttle/missions/sts-68/news/sts-68-mcc-05.txt HTTP/1.0"` is the request line from the client. The first part is the HTTP method (GET), followed by the requested resource (endpoint) and the HTTP version.
- `200` is the HTTP status code indicating that the request was successful.
- `1839` is the size of the response in bytes.

Using xargs to process the log file

You can use `xargs` to process the log file in various ways.

Example: Extracting unique IP addresses

Here's how you can extract unique IP addresses from the log file using `xargs`:

```
find . -name "NASA_access_log_Aug95" | xargs awk '{print $1}' | sort | uniq
> unique_ips.txt
```

Example output of the first few lines of `unique_ips.txt`:

```
    001.msy4.communique.net
    007.thegap.com
    01-dynamic-c.wokingham.luna.net
    01.ts01.zircon.net.au
    02-17-05.comsvc.calpoly.edu
```

This command finds the log file, uses `awk` to print the first field (IP addresses), sorts them, and then uses `uniq` to filter out duplicate entries.

Example: Counting requests per IP address

Here's how you can count the number of requests per IP address using `xargs`:

```
find . -name "NASA_access_log_Aug95" | xargs awk '{print $1}' | sort | uniq
-c | sort -nr > ip_request_counts.txt
```

Example output of the first few lines of `ip_request_counts.txt`:

```
    6530 edams.ksc.nasa.gov
    4846 piweba4y.prodigy.com
    4791 163.206.89.4
    4607 piweba5y.prodigy.com
    4416 piweba3y.prodigy.com
```

In this example, we use `uniq -c` to count the occurrences of each IP address, and then sort the results numerically in reverse order to see the most frequent requesters at the top.

Example: Filtering requests by status code

404 - Not Found Status Code

You can filter requests by a specific HTTP status code using `xargs`. For example, to find all requests that resulted in a 404 error:

```
find . -name "NASA_access_log_Aug95" | xargs grep '404 -' > 404_errors.txt
```

Example output of the first few lines of `404_errors.txt`:

```
     js002.cc.utsunomiya-u.ac.jp - - [01/Aug/1995:00:07:33 -0400] "GET
/shuttle/resources/orbiters/discovery.gif HTTP/1.0" 404 -
     tia1.eskimo.com - - [01/Aug/1995:00:28:41 -0400] "GET
/pub/winvn/release.txt HTTP/1.0" 404 -
     grimnet23.idirect.com - - [01/Aug/1995:00:50:12 -0400] "GET
/www/software/winvn/winvn.html HTTP/1.0" 404 -
     miriworld.its.unimelb.edu.au - - [01/Aug/1995:01:04:54 -0400] "GET
/history/history.htm HTTP/1.0" 404 -
     ras38.srv.net - - [01/Aug/1995:01:05:14 -0400] "GET
/elv/DELTA/uncons.htm HTTP/1.0" 404 -
```

200 - Success Status Code

You can filter requests by a specific HTTP status code using `xargs`. For example, to find all requests that resulted in a 200 success status:

```
find . -name "NASA_access_log_Aug95" | xargs grep ' 200 ' > 200_success.txt
```

Example output of the first few lines of `200_success.txt`:

```
     in24.inetnebr.com - - [01/Aug/1995:00:00:01 -0400] "GET
/shuttle/missions/sts-68/news/sts-68-mcc-05.txt HTTP/1.0" 200 1839
     ix-esc-ca2-07.ix.netcom.com - - [01/Aug/1995:00:00:09 -0400] "GET
/images/launch-logo.gif HTTP/1.0" 200 1713
     slppp6.intermind.net - - [01/Aug/1995:00:00:10 -0400] "GET
/history/skylab/skylab.html HTTP/1.0" 200 1687
     piweba4y.prodigy.com - - [01/Aug/1995:00:00:10 -0400] "GET
/images/launchmedium.gif HTTP/1.0" 200 11853
     slppp6.intermind.net - - [01/Aug/1995:00:00:11 -0400] "GET
/history/skylab/skylab-small.gif HTTP/1.0" 200 9202
```

Example: Count lines in the file

You can count the total number of lines in the log file using `xargs` as follows:

```
find . -name "NASA_access_log_Aug95" | xargs wc -l
```

Example output:

```
1569898 ./NASA_access_log_Aug95
```

Conclusion

Using `xargs` in combination with commands like `find`, `grep`, and `awk` can significantly enhance your ability to process and analyze large datasets, such as log files. It allows for more efficient command execution and helps overcome limitations related to command line length. The examples provided demonstrate various ways to extract and analyze information from the NASA access log file, showcasing the versatility of `xargs` in real-world scenarios.