Log Analysis 101

When searching through large volumes of logs, doing it by hand is not a feasible solution. Searching through large amounts of text is something computers do much better than we can, if you know how to ask the right questions. Using a few command line tools, we can quickly manipulate these log files to give us answers to our questions.

My Frequently Used tools

- grep
- sort
- uniq
- wc
- cut

grep

You can provide grep either a string literal or a regular expression, and grep returns each line of text that contains that text.

Of the commands I use grep is the most versatile and complex. There will be more examples later on.

Here grep is finding each line of the grep manual page containing the word grep

```
ubuntu@DESKTOP-V48DOKI:/mnt/c/Users/Noah/Desktop$ man grep | grep grep
       grep, egrep, fgrep, rgrep - print lines matching a pattern
       grep [OPTIONS] PATTERN [FILE...]
       grep [OPTIONS] -e PATTERN ... [FILE...]
       grep [OPTIONS] -f FILE ... [FILE...]
       grep  searches  for  PATTERN in each FILE.  A FILE of “-” stands for standard input.  If no FILE is
       input.  By default, grep prints the matching lines.
       In  addition,  the  variant  programs  egrep, fgrep and rgrep are the same as grep -E, grep -F, and
       grep -r, respectively.  These variants are deprecated, but are provided for backward compatibility.
              Output the version number of grep and exit.
              and grep -P may warn of unimplemented features.
              file,  and NUM matching lines are output, grep ensures that the standard input is positioned
```

When using command line tools you can toggle extra commands and supply more input.

These are the flags for grep I use most commonly.

Flag	Description
-v	This is an inverse search. Instead of returning each line of text that matches, it returns only lines of text NOT matching the given text.
-o	This is an only-matching flag. Instead of returning the whole line of text, it returns ONLY the matching text.
-E	This flag allows you to search for a regular Expressions, which will be covered in later examples.

sort

Sort is a straight forward command that sorts all lines of text and returns them in alphabetic order.

The only “gotcha” you may run into with sort is how it deals with sorting numbers.

This is what happens when you try to sort numbers

Unsorted List	Sorted
111	1
123	111
223	123
2	2
1	223
321	3
333	321
3	333

In the previous example you could see that running sort on this list correctly sorts them alphabetically, but numerically incorrect.
Using sort –n will correctly sort numbers

Flag	Description
-n	This flag will sort the list as expected when dealing with numbers

Unsorted List	Sorted with `sort -n`
111	1
123	2
223	3
2	111
1	123
321	223
333	321
3	333

uniq

uniq short for unique. This command will collapse any repeating lines of text as seen in bellow.

Before uniq	After uniq
Repeated 1	Repeated 1
Repeated 1	Repeated 2
Repeated 1	Repeated 3
Repeated 2	
Repeated 2	
Repeated 2	
Repeated 3	
Repeated 3	

Flag	Description
-c	This flag will display the count of how many lines were collapsed.

Before uniq	After `uniq -c`
Repeated 1	3 Repeated 1
Repeated 1	4 Repeated 2
Repeated 1	2 Repeated 3
Repeated 2	
Repeated 2	
Repeated 2	
Repeated 2	
Repeated 3	
Repeated 3	

WC

The most straight forward of the group, wc is short for Word Count. When used without any flags wc will display number of lines, number of characters, file size. This is a very useful command for any question that starts with “How many…”

Flag	Description
-l (lowercase L)	Displays only the number of lines

Bringing it all together with | (pipe)

In the command line you may often want to run more than one command to get your results.

The | character is called a “pipe” because you take the output from the command on the left, and “pipe” it into the next command as input. It is usually found on the top right area of your keyboard (shift + \)

Example.txt

http 500 Internal Server Error
http 301 Moved Permanently
http 418 I'm a teapot
http 403 Forbidden
http 502 Not Implemented
http 404 Not Found
http 407 Proxy Authentication Required
http 200 OK
http 503 Service Unavailable
http 401 Unauthorized

grep “http 4” Example.txt

```
http 418 I'm a teapot
http 403 Forbidden
http 404 Not Found
http 407 Proxy Authentication Required
http 401 Unauthorized
```

There are some more things you might want to do to this text, lets see how to take this data and sort it within the same command.

grep “http 4” Example.txt | sort

http 401 Unauthorized

http 403 Forbidden

http 404 Not Found

http 407 Proxy Authentication Required

http 418 I'm a teapot

What we did here was take the first output of grep, pipe it to sort, which took that data and did it’s operation, then returning the sorted values of each line containing “http 4”.

This can be done with any commands that output text. We just as easily could have taken our grep and piped it to wc -l for the answer of 5

Pipe level 2

Example2.txt

Good 1
Bad 1
Good 2
Good 1
Good 1
Good 2
Good 3
Bad 1
Bad 2
Good 2
Bad 1
Good 1

Now we want to see only the unique values of this list. Lets see what happens if we pipe this to uniq

```
grep “Good” Example2.txt
```

Good 1

Good 2

Good 1

Good 1

Good 2

Good 3

Good 2

Good 1

```
grep “Good” Example2.txt | uniq
```

```
Good 1
Good 2
Good 1
Good 2
Good 3
Good 2
Good 1
```

Why didn’t we see a list of only one Good 1, Good 2, and Good 3?

Remember, uniq only collapses lines of text that are the same, and adjacent to each other. What we need to do is sort the list first, then we can find all the uniq values.

grep “Good” Example2.txt | sort | uniq

Good 1
Good 2
Good 3

1. We find all lines of text with Good
2. We sort them so all the similar lines are adjacent
3. we take the sorted data and run it through uniq to find the unique values

Now what if they asked which line was repeated the most? In this example it would be easy to just count it, but what if there were 100,000 lines of text with 1000 possible values?

To get that answer, we will use uniq -c which shows how many times a line of text was collapsed.

```
grep “Good” Example2.txt | sort | uniq -c | sort -n
4   Good 1
3   Good 2
1   Good 3
```

Regular Expressions & grep

Regular expressions (or regex) are patterns used to match character combinations in strings. When we want to use regex with grep we have to use the -E flag. In NCL problems, the most common use of regular expressions is searching for IP addresses.

Q: How many IP addresses failed to log in?

A: grep “authentication failed” logs.txt | grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" | sort | uniq | wc -l

Regular expressions can be somewhat confusing and takes time to become familiar with them. Lets break down what we are seeing in this example

```
grep “authentication failed” logs.txt | grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" | sort | uniq | wc -l
```

```
grep “authentication failed” logs.txt
```

Gets each line of text with “authentication failed” from logs.txt

```
grep “authentication failed” logs.txt | grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" | sort | uniq | wc –l
```


```
grep “authentication failed” logs.txt | grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" | sort | uniq | wc -l
```


- `| grep` Take the output from the first grep, and pipe it to another grep
- `-oE` Use the only-matching, and regular expression flags
- `\b` \b has a special meaning in regex meaning “word boundary”.
- `([0-9]{1,3}\.)` This means “I want to match any character 0-9 (`[0-9]`), and I want it to repeat 1, 2, or 3 (`{1,3}`) times, and after those 1, 2, or 3 numbers I want a . (period character). Just like how \b had a special meaning, a . (period character) has a special meaning. When you want to denote a period character you have to “escape” it with a leading \
- `{3}` This means take the 1-3 numbers followed by a period, and repeat that 3 times.
- `[0-9]{1,3}` Just like before, we want to match any numbers 1-3 times.
- `| sort | uniq | wc -l` Take the IP addresses from the previous grep command, sort it, get the unique values, and give us a count of how many there were.

Regular expressions have a whole set of special characters. This is just something you will get used to with more practice. To learn more about using regex visit regexone.com which has levels you will progress through.

My favorite site when I want to test a regular expression is regex101.com.

HAPPY HUNTING!