Trent University

COIS3380H
Winter 2023
Lab 1
Due: January 26th, 2023 (11:59pm)

It is strongly advised that, for each lab, you create an individual directory on loki.trentu.ca to do your work. That way when it comes time to submit your lab, you need only backup the content of that specific directory.

You could use the following command to create and change to a work area for lab #1.

```
username:~> mkdir lab1
username:~> cd lab1
username:lab1>
```

Using the notes provided on Blackboard for shell scripting, create solutions for the following three problems. Your solution will be in the form of a **properly documented** shell script that can be run on demand. I will be posting a requirements document for in-code documentation to ensure we are all of the same page for what "properly documented" means.

You will need to use the ***grep***, ***awk*** and other commands discussed in class to be able to complete this lab. All of the required components have been presented in class and are in the notes. You need only shuffle their order or tweak the examples to make these two tasks work for you. you can always use the ***man*** pages to get additional details for each command.

NOTE: You will be required to submit all of your lab code as a zip file through blackboard. This is true for all labs submitted this semester. Though you can develop your code on other platforms if you so wish, ONLY the version of your code **on the course server** will be used for grading.

IF YOUR CODE IS NOT THERE, THERE'S NOTHING TO GRADE !!!

NOTE: It is suggested that you create a directory in you $HOME directory for each course. In this case you would create a directory named COIS3380 or just 3380. For each lab in the course, create a separate directory: lab1, lab2..lab4. This will make finding your work on Loki a lot easier when it comes time to grade assignments. It will also help you with this first assignment. **DO NOT** create your 3380 directory inside your public_html directory. It will expose all of your work to the internet and promote academic integrity issues. You are responsible for securing your work from copying.

1) For this portion of the lab, you are asked to create an "assignment submission tool" as a shell script. The script will create a ZIP file for you, of the contents of the directory you specified on the command line. The directory you will specify will be your current assignment directory. This ZIP file is what you will submit to blackboard for grading. See below for a quick description of the zip command line parameters.

You are required to write the script in such a fashion to have it create the zip file for you using the proper naming convention. If you recall from the syllabus, I require that all submissions be in the form: LastName_FistName_DirectoryName.zip. Since your login name on the server is unique, and consists of firstnamelastname, you may use the environment variable $USER (or $LOGNAME) as an alternative if you so desire. Should you decide to use FIRSTNAME and LASTNAME, there are no environment variables which individually contain your first or last names. You will need to create these variables within your script (should you choose the proper naming convention). It would be considered "bad form" to hardcode your names directly into your commands. Variable names are preferred just in case you move this script to another machine which uses a different naming convention.

You are expected to use variables for these and NOT hard code them into the ZIP filename. Of course, if you want, you could create the environment variables yourself in your .bashrc (but then you would have to test for their initialization in the script to not wind up with NULLs).

Your script should accept a command line argument. This should be the **name** of the sub-directory you are going to ZIP-up for submission. Command line parameters are accessible within your script as $1, $2, $3...

To prevent issues when unpacking everyone's work for grading, it is **important** that your ZIP command include the **full path** to the target sub-directory. To accomplish this you MUST use the environment variable **`$HOME`** as part of the ZIP command line. Make sure that your script creates the ZIP file in the HOME directory of your account. If it creates it in the directory you are trying to ZIP up, the ZIP file itself will be corrupted. You may find the $PWD environment variable useful in this endeavour.

Make sure the script is properly documented, is executable and that you identify it, using the established UNIX convention discussed in class as well as the proper shell interpreter it is intended to run under.

How to use ZIP: (see the man page for more details)

```
  zip    zipfilename       [list of files or directories to include in the zipfile]
```

```
e.g.: jacques@UBU64vm:~$ zip -r  jacquesabeland_lab1.zip     /home/jacquesabeland/lab1
                            verb     LOGNAME    1st parameter    HOME        1st parameter
```

the -r flag for the zip command causes the program to RECURSE into subdirectories should they exist. Omitting this flag will result in a ZIP file that only contains the "directory" file and not its contents. Remember that in UNIX everything is a FILE. Unless you tell it, zip will only backup the file you specified (the directory name).

If you omit the -r, or choose not to use it, make sure you append a "/*" to the end of the source directory name to include all of the files contained within the directory.

Once you have created the ZIP file, **test it** to make sure it contains the contents you expect BEFORE uploading it to Blackboard. An empty submission "on time" does not circumvent late penalties for a secondary submission that is late.

You can test your ZIP by listing the contents: unzip -l ZIP_filename.zip

Question 2)

For this question, please refer to the "Unix Shell Programming" notes from class. Some of the slides have relevant examples you can use as reference.

Large Datasets: As a system's admin, you are often called upon to find the source of a problem. Sometimes, the issue can be security related and you are required to find out who's been hacking at your system. In this portion of the lab you are being asked to create a shell script to analyze your WEB logs. You are looking for people who are trying to break into your web server. The site only hosts two valid web pages: "/" is for the default web site for the server and /~forensics" is the one user page. All other log entries pointing to other pages, are signs of hacking.

go in the folder and do ls to find an access log file for yourself?

In this lab we will look at the "access" logs in /home/COIS/3380/lab1. There are over 1600 of them! Entries will look something like:

```
[root@loki lab1]# head -n 5 system_logs_backup_2019-07-29_access.log.12
35.203.128.47 - - [16/Jul/2019:06:39:40 -0400] "OPTIONS / HTTP/1.1" 200 181 "-" "-"
170.82.21.20 - - [16/Jul/2019:08:44:38 -0400] "GET / HTTP/1.1" 200 1654 "-" "Mozilla/5.0 (Windows
NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36"
40.77.167.19 - - [16/Jul/2019:08:51:24 -0400] "GET / HTTP/1.1" 200 1081 "-" "Mozilla/5.0
(compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)"
181.210.55.178 - - [16/Jul/2019:08:54:19 -0400] "GET / HTTP/1.1" 200 1654 "-" "Mozilla/5.0
(Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2
Safari/601.7.7"
80.82.70.118 - - [16/Jul/2019:09:32:23 -0400] "GET / HTTP/1.0" 200 1673 "-" "masscan/1.0
(https://github.com/robertdavidgraham/masscan)"
```

(Notice: most lines wrap, others don't).

the script should take access log file as an argument

Part 1: Using the regular expression given in class as examples, find all of logged requests for non-existing pages. Typically in a web log, the error number for a non-existing page is " 404 " (notice the spaces before and after the 404!). Extract and count the unique page references.

NOTE: For the purpose of testing and debugging your script, PLEASE WORK WITH A SINGLE LOG FILE: /home/COIS/3380/lab1. Pick one that will generate some valid test output as you work. If you pick one that is too small, you may not get anything from your script. Using a single file for testing will alleviate a significant load from the server and allow everyone to work concurrently.

The required output will be two fold. First, a file that contains ALL of the matches you script generated and the count of the number of times they were seen in the logs. You can call this file whatever you want but choose an appropriate name. The second file will contain only the first 10 lines top ten (10) most-requested invalid pages (use the full output file to generate this one). Your second output should be: top_ten_404_pages.txt.

Without giving out the commands, your output should look something like this should you test with the file: /home/COIS/3380/lab1/system_logs_backup_2019-11-04_access.log.3. Just the first 5 lines of output:

```
          27 "POST /test.php
```

```
24 "POST /1.php
18 "POST /qq.php
12 "POST /x.php
12 "POST /ss.php
```

config.php
api.php
robots.txt

Some of the failed page requests are for know utilities such as phpMyAdmin. Some of the other failed requests are "muhstik", "cacti" and "broadworks". Take 3 of the pages that look weird and Google them to see what nefarious things the hackers are trying to exploit. Include them in your report. If Google returns nothing, pick another one.

For your final testing and submission however, ensure your script is modified **to use all of the appropriate logs** in the target directory. Better yet, if you want to try it, use a command line parameter and pass the filename to your script!

For this exercise, the logs can be found in /home/COIS/3380/lab1/ with a filename starting with the word "system_logs". For your final submission, **You must use all of the logs provided** and create a single summary report. All of the logs combined account for just over 614,474 lines of messages.

Hint: if you pass the file specification as a parameter to your script and the specification includes a wildcard, the parameter must be **enclosed in quotes on the command line (not in the script).**

```
grep -e " 404 " /home/COIS/3380/lab1/system_logs* | awk '{print $6, $7}' | sort |
uniq -c | sort -nr | head -n 10 > /home/punyajamishra/3380/lab1/lab1_countedlog.txt
```

Part 2) Now go back to the original set of data files and extract all of the IP addresses which are looking for invalid pages. Let's find out where the **top two** ip addresses come from. Similar to the previous exercise, find all of the " 404 " page requests and extract the IP address from each record. Produce an output file which contains only the top 10 most frequent IP addresses and store that output into a file: top_10_IPs.txt.

For testing should you use the file :

```
grep -e " 404 " system_logs_backup_2019-07-29_access.log.3 | grep -o '
[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}' | sort | uniq -c | sort -nr
```

/home/COIS/3380/lab1/system_logs_backup_2019-11-04_access.log.3

Then you should get something like:

```
892 192.169.231.213
892 106.12.123.186
891 101.51.106.223
  7 106.13.116.111
  6 120.92.123.150
  3 157.55.39.245
  2 46.4.65.7
      ...
```

Manually, take the two most frequent IP addresses and find out where they are from. You can pass the IP address to the web site http://ipinfo.io/. It will then return the geolocation of the IP (as best it can). Writing the code for it (leaning heavily on the examples provided) would be preferable. No difference in grading for either approach.

To get the geolocation, pass the IP address as if it were a page reference in the HTTP.

```
wget http://ipinfo.io/192.169.231.213 -o /home/punyajamishra/3380/lab1/192.169.231.213
```

e.g.

```
[jacques@loki lab1]$ wget http://ipinfo.io/192.169.231.213
--2023-01-12 09:24:20--  http://ipinfo.io/192.169.231.213
Resolving ipinfo.io (ipinfo.io)... 34.117.59.81
Connecting to ipinfo.io (ipinfo.io)|34.117.59.81|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 313 [application/json]
Saving to: '192.169.231.213'

100%[==========================================================================>] 313
--.-K/s   in 0s

2023-01-12 09:24:20 (29.2 MB/s) - '192.169.231.213' saved [313/313]
[jacques@loki lab1]$ cat 106.12.123.186
{
  "ip": "106.12.123.186",
  "city": "Shenzhen",
  "region": "Guangdong",
  "country": "CN",
  "loc": "22.5455,114.0683",
  "org": "AS38365 Beijing Baidu Netcom Science and Technology Co., Ltd.",
  "timezone": "Asia/Shanghai",
  "readme": "https://ipinfo.io/missingauth"
}
```

```
[jacques@loki lab1]$ wget http://ipinfo.io/106.12.123.186
--2023-01-12 09:24:29--  http://ipinfo.io/106.12.123.186
Resolving ipinfo.io (ipinfo.io)... 34.117.59.81
Connecting to ipinfo.io (ipinfo.io)|34.117.59.81|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 273 [application/json]
Saving to: '106.12.123.186'

100%[===========================================================================>] 273
--.-K/s   in 0s

2023-01-12 09:24:29 (9.88 MB/s) - '106.12.123.186' saved [273/273]
[jacques@loki lab1]$ cat 192.169.231.213
{
  "ip": "192.169.231.213",
  "hostname": "213.231.169.192.host.secureserver.net",
  "city": "Phoenix",
  "region": "Arizona",
  "country": "US",
  "loc": "33.4484,-112.0740",
  "org": "AS26496 GoDaddy.com, LLC",
  "postal": "85001",
  "timezone": "America/Phoenix",
  "readme": "https://ipinfo.io/missingauth"
}
```

Other geolocation tools you might want to use:

This one accepts IP addresses as part of the URL:

https://tools.keycdn.com/geo

e.g. https://tools.keycdn.com/geo?host= 192.249.185.20

This tool is manual:

https://www.home.neustar/resources/tools/ip-geolocation-lookup-tool

Submitting your work:

After you have completed the two scripts and geolocated the source, ZIP up your Lab1 directory and submit it to Blackboard. Ensure you ZIP file contains data and that it is named properly. You can use the "unzip -**ℓ**" command to get a listing of the contents of the ZIP. That's a lower case L and not a one(1).

e.g. (Your filenames may differ from those shown)

```
[root@loki 3380]# unzip -l /home/jacques/jacques_beland_lab1.zip
Archive:  /home/jacques/jacques_beland_lab1.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
      163  09-21-2017 09:49   home/jacques/3380/lab1/58.242.83.7
      163  09-21-2017 09:49   home/jacques/3380/lab1/58.242.83.7.1
     3965  09-21-2017 09:49   home/jacques/3380/lab1/lab1_logfile.txt
      130  07-12-2017 11:38   home/jacques/3380/lab1/find_top_10.sh
  3464447  08-12-2017 09:59   home/jacques/3380/lab1/lab0_ip_addresses.txt
---------                     -------
  3464903                     5 files
[root@loki 3380]#
```

You should see all of your scripts as well as the files created for the two IP locate commands. Notice that the FULL PATH to the files is shown in the listing!!!

Logging your work:

The log file that you are required to include should be created AFTER you have done all of your coding and testing. That will keep your log file as clean as possible.

The log should include at the very least:
1) you running your code on the single target log file
2) running your code on all of them together (all of the log files "*")
3) a geolocation on the top two candidates from part 2

You should then place your log file in your assignment1 or lab1 directory and then, and only then, create the zip file to be submitted to Blackboard.

Note: if you use the Unix script command to create your logs, you MUST clean them up before submission.

https://superuser.com/questions/236930/how-to-clean-up-output-of-linux-script-command/650877
https://www.thegeekstuff.com/2011/10/grep-or-and-not-operators/