**Linux administration with bash. Home task**

**A. Create a script that uses the following keys:**

1. When starting without parameters, it will display a list of possible keys and their description.

2. The --all key displays the IP addresses and symbolic names of all hosts in the current subnet

3. The --target key displays a list of open system TCP ports.

The code that performs the functionality of each of the subtasks must be placed in a separate function

Here is our script:

#!/bin/bash

# function to display list of possible keys and their description

display\_keys() {

echo "Possible keys and their description:"

echo "--all: Displays the IP addresses and symbolic names of all hosts in the current subnet"

echo "--target: Displays a list of open system TCP ports"

}

# function to display IP addresses and symbolic names of all hosts in the current subnet

display\_all() {

echo "IP addresses and symbolic names of all hosts in the current subnet:"

arp -a

}

# function to display a list of open system TCP ports

display\_target() {

echo "List of open system TCP ports:"

netstat -tuln | grep -E '^tcp'

}

if [ $# -eq 0 ]; then

display\_keys

else

while [ $# -gt 0 ]; do

case $1 in

--all)

display\_all

shift

;;

--target)

display\_target

shift

;;

\*)

echo "Invalid key. Use --help for a list of possible keys"

exit 1

;;

esac

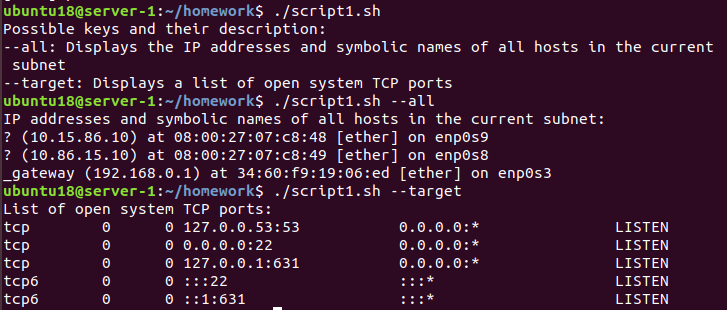
done

fi

This script starts by defining three functions: display\_keys, display\_all, and display\_target. The first function will display the possible keys and their descriptions when the script is run without any parameters. The second function will use the arp command to display the IP addresses and symbolic names of all hosts in the current subnet. The third function will use the netstat command to display a list of open system TCP ports.

Then the script checks if any parameter has been passed or not. If no parameter passed it calls the display\_keys function else it uses a while loop to iterate over the passed parameters, and a case statement is used to check if the parameter is --all or --target and calls the corresponding function.

Here is the output on my VM:



**B. Using Apache log example create a script to answer the following questions:**

1. From which ip were the most requests?

2. What is the most requested page?

3. How many requests were there from each ip?

4. What non-existent pages were clients referred to?

5. What time did site get the most requests?

6. What search bots have accessed the site? (UA + IP)

Here is our script:

#!/bin/bash

# set the log file location

log\_file='/home/ubuntu18/homework/example\_log.log'

# 1. From which ip were the most requests?

echo "IP address with the most requests:"

grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" $log\_file | sort | uniq -c | sort -nr | head -n 1

# 2. What is the most requested page?

echo "Most requested page:"

grep -oE "\"[A-Z]{3,4} [^\"]+" $log\_file | awk '{print $2}' | sort | uniq -c | sort -nr | head -n 1

# 3. How many requests were there from each ip?

echo "Number of requests from each IP:"

grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" $log\_file | sort | uniq -c | sort -nr

# 4. What non-existent pages were clients referred to?

echo "Non-existent pages:"

grep "404" $log\_file | awk '{print $7}' | sort | uniq -c | sort -nr

# 5. What time did site get the most requests?

echo "Time with the most requests:"

grep -oE "[[:digit:]]{2}/[[:alpha:]]{3}/[[:digit:]]{4}:[[:digit:]]{2}:[[:digit:]]{2}" $log\_file | awk -F: '{print $2}' | sort | uniq -c | sort -nr | head -n 1

# 6. What search bots have accessed the site?

echo "Search bots that accessed the site (UA + IP):"

while read line; do

if [[ $line =~ (bot|crawler|spider) ]]; then

ip=$(echo $line | awk '{print $1}')

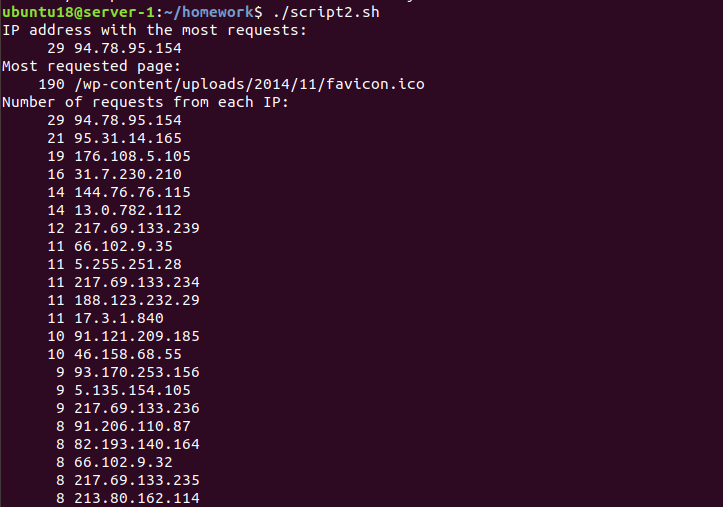
ua=$(echo $line | awk -F\" '{print $6}')

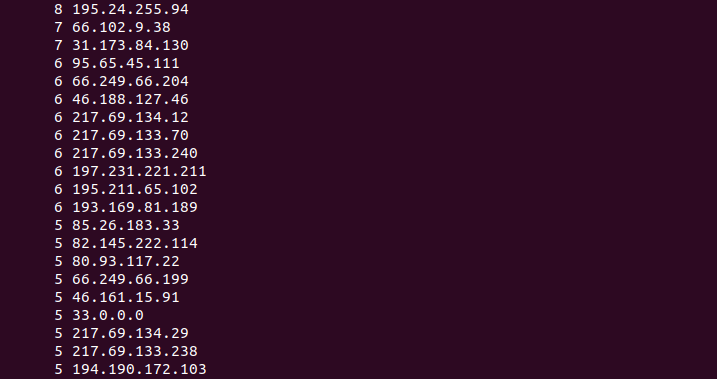
echo "IP: $ip, UA: $ua"

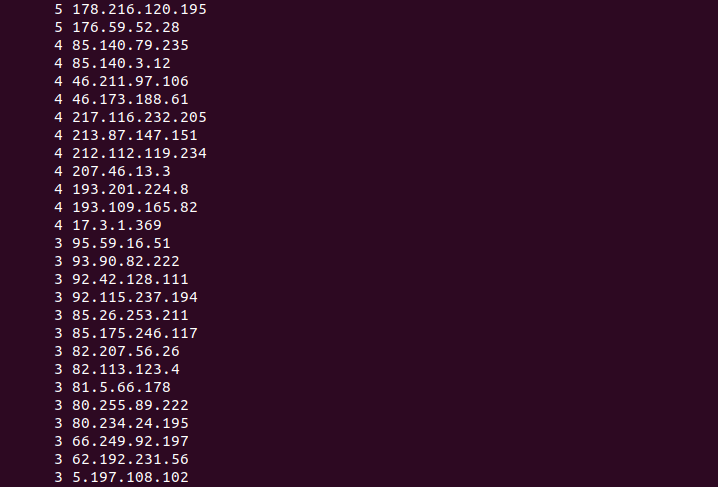
fi

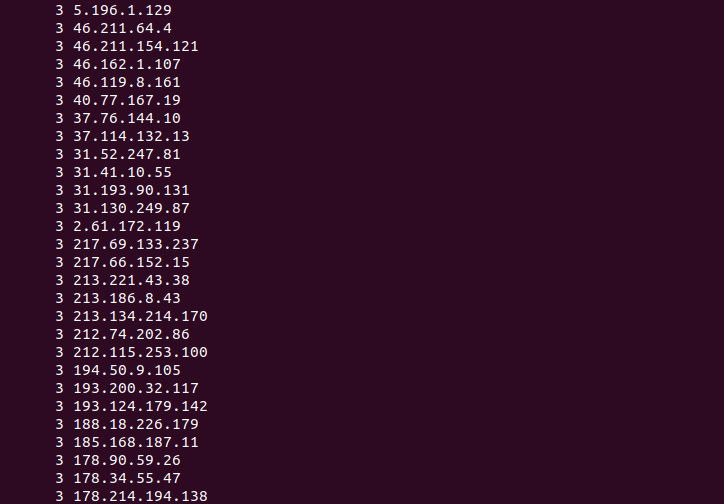
done < <(grep -oE "[^ ]+\"[^\"]+\"" $log\_file)

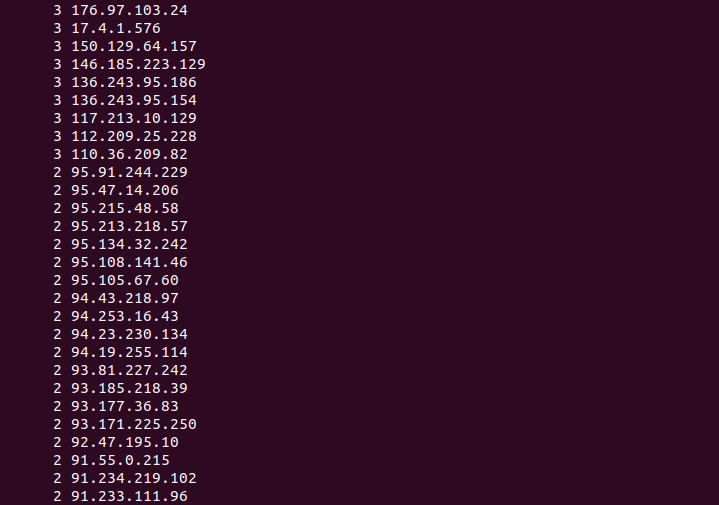
1. To find the IP address with the most requests, we can use *grep* to extract all the lines containing IP addresses, then use *sort* to sort the lines by IP address, and finally use *uniq* with the *-c* option to count the number of occurrences of each IP. The IP address with the highest count will be the one that made the most requests.
2. To find the most requested page, we can use *grep* to extract all the lines containing the request method and the requested page, then use *sort* to sort the lines by page, and finally use *uniq* with the *-c* option to count the number of occurrences of each page. The page with the highest count will be the most requested.
3. To find the number of requests from each IP, we can use the same method as in question 1, but instead of sorting by IP, we can sort by the count number.
4. To find non-existent pages, we can use *grep* to extract all lines containing the status code "404" and the requested page.
5. To find the time with the most requests, we can use *grep* to extract all lines containing the timestamp, then use *awk* to extract the hour or the minute from the timestamp, then use *sort* to sort the extracted values and finally use *uniq* with the *-c* option to count the number of occurrences of each hour or minute. The hour or minute with the highest count will be the one that had the most requests.
6. This script uses a while loop to read each line of the log file, and it uses the *grep* command to extract the IP and user agent from each line. The script then uses the *awk* command to extract the IP and user agent from the line, and it uses an if statement to check if the user agent contains the word "bot", "crawler", or "spider". If the user agent contains one of these words, the script will print the IP and user agent of that search bot.



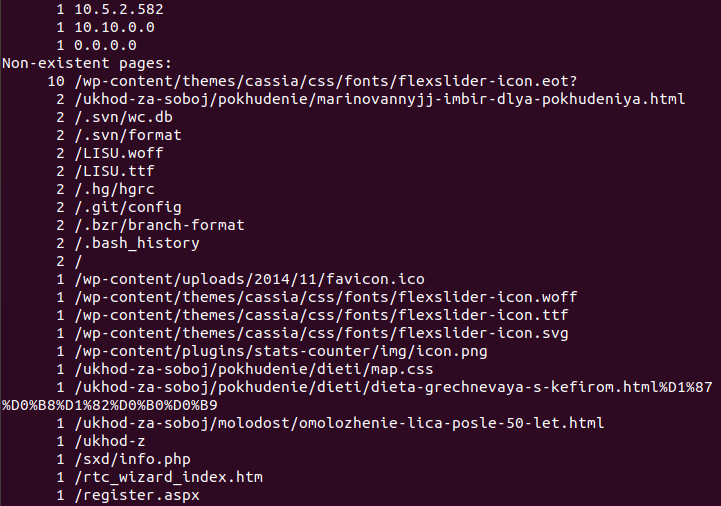


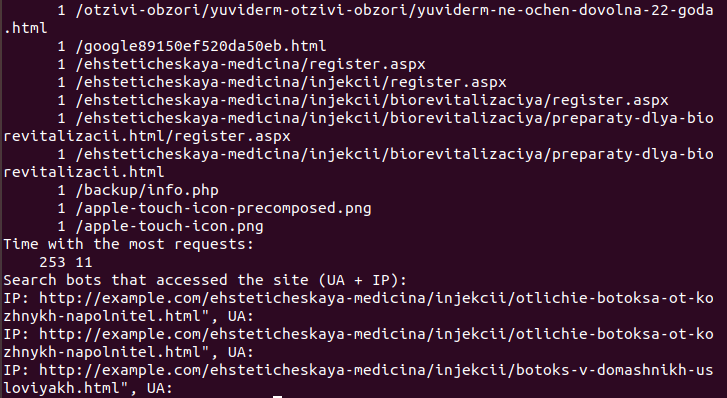






A long list with the number of requests from other ips…





**C. Create a data backup script that takes the following data as parameters:**

1. Path to the syncing directory.

2. The path to the directory where the copies of the files will be stored.

In case of adding new or deleting old files, the script must add a corresponding entry to the log file

indicating the time, type of operation and file name. [The command to run the script must be added to

crontab with a run frequency of one minute

Here is our script:

#!/bin/bash

# Get the current date and time

current\_time=$(date +%Y-%m-%d\_%H:%M:%S)

# Syncing directory path

sync\_dir=$1

# Backup directory path

backup\_dir=$2

# Log file path

log\_file=backup.log

# Copy all files from the syncing directory to the backup directory

rsync -av $sync\_dir $backup\_dir --ignore-existing

# Find any new or deleted files in the syncing directory

new\_files=$(find $sync\_dir -type f -cnewer $backup\_dir)

for file in $new\_files; do

echo "$current\_time: Added $file" >> $log\_file

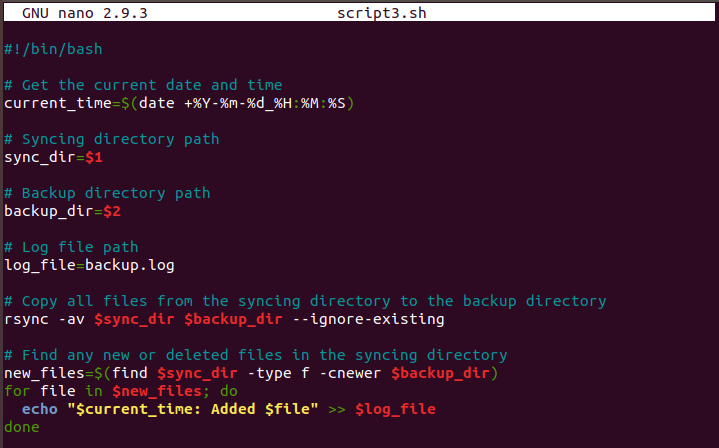
done

deleted\_files=$(find $backup\_dir -type f ! -newer $sync\_dir)

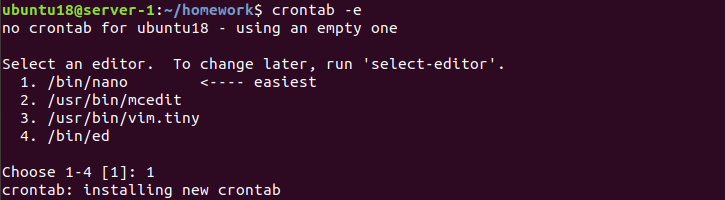
for file in $deleted\_files; do

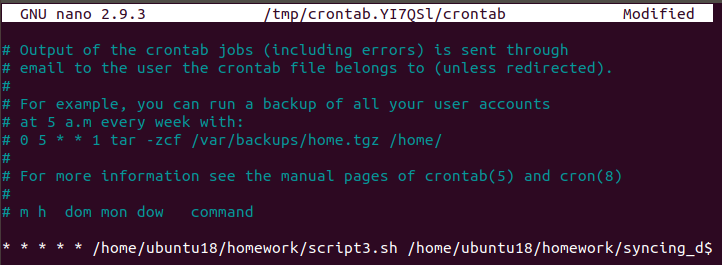
echo "$current\_time: Deleted $file" >> $log\_file

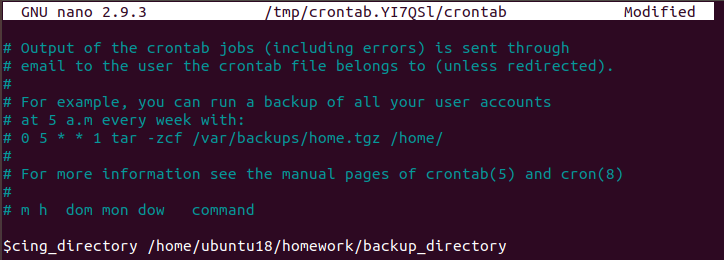
done

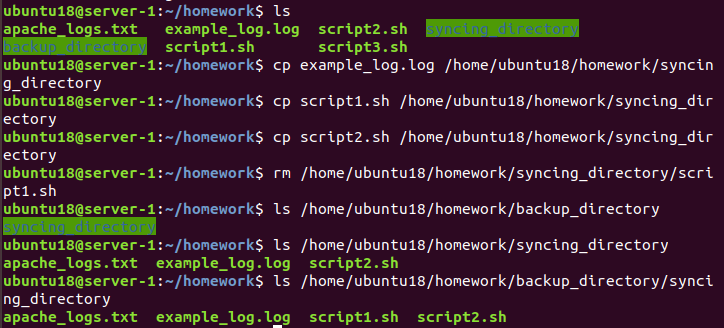


Let`s open the crontab file and add the task:







Let`s test the execution of our script:

Let`s check the log file:

