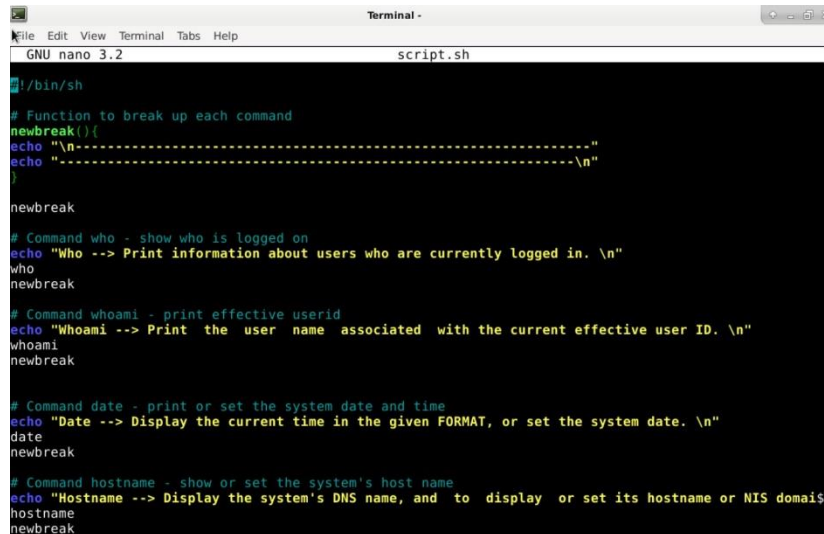


CS 446 Homework 1

Chapter 1

1. [Linux Operating System] A shell script is a sequence of shell commands written in an executable script file. Executing this file instructs the shell to execute all commands in the order of their appearance in the script file. There are several shell scripting tutorials available on the web, e.g. search by entering the keywords *Linux shell script tutorials*. Go through one of these tutorials and then write a shell script that displays various system parameters by using shell commands like *who*, *whoami*, *date*, *hostname*, etc.

- Code attached as ‘ScriptQ1.sh’
- Script Source Code



```

GNU nano 3.2 script.sh

#!/bin/sh

# Function to break up each command
newbreak(){
echo "\n-----\n"
}

newbreak

# Command who - show who is logged on
echo "Who --> Print information about users who are currently logged in. \n"
who
newbreak

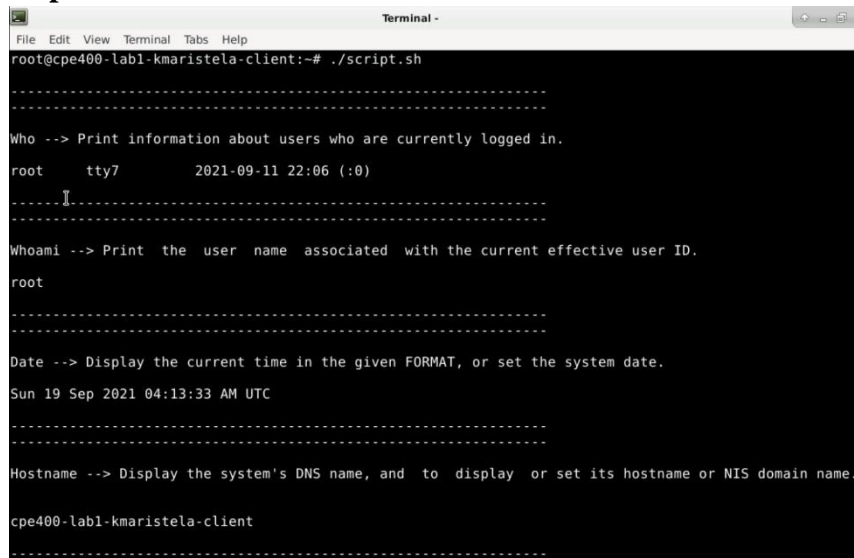
# Command whoami - print effective userid
echo "Whoami --> Print the user name associated with the current effective user ID. \n"
whoami
newbreak

# Command date - print or set the system date and time
echo "Date --> Display the current time in the given FORMAT, or set the system date. \n"
date
newbreak

# Command hostname - show or set the system's host name
echo "Hostname --> Display the system's DNS name, and to display or set its hostname or NIS domain name. \n"
hostname
newbreak

```

- Script Output



```

root@cpe400-lab1-kmaristela-client:~# ./script.sh

-----

Who --> Print information about users who are currently logged in.
root    tty7        2021-09-11 22:06 (:0)
-----

Whoami --> Print the user name associated with the current effective user ID.
root
-----

Date --> Display the current time in the given FORMAT, or set the system date.
Sun 19 Sep 2021 04:13:33 AM UTC
-----

Hostname --> Display the system's DNS name, and to display or set its hostname or NIS domain name.
cpe400-lab1-kmaristela-client
-----

```

Chapter 2

2. [Linux Operating System] **ps** is a command that displays information about all processes currently running in your system. Read man page of **ps** command. Enter the following commands: (1) **ps -ef / more** and (2) **ps -aux / more**. Both of these will result in displaying a long list of processes. Identify what processes are started when the system is booted, and what processes are started later on. For each process, find out who owns it, what code it is running, and how much CPU/memory it has used.

Now, store the details of all processes owned by root in a file called *rootprocesses-1*, and all processes owned by you in a file called *my-processes-1*.

Next, restart your system, and create similar files, *root-processes-2* and *myprocesses-2*. Compare *root-processes-2* with *root-processes-1*, and *my-processes-2* with *my-processes-1*. Explain the differences between the two.

- **Processes saved in folder 'Q2'**
- **Process Column Explanation**
 - **USER:** Effective user ID; user that executed the process
 - **PID:** Number representing the process ID
 - **%CPU:** CPU utilization of the process
 - **%MEM:** Ratio of the process's resident set size to the physical memory on the machine
 - **VSZ:** Virtual memory size of the process in KiB (1024-byte units)
 - **RSS:** Resident set size; non-swapped physical memory that a task has used
 - **TTY:** Controlling TTY (terminal)
 - **STAT:** Multi-character process state
 - **START:** Starting time or date of the process
 - **TIME:** Cumulative CPU time
 - **COMMAND:** command with all its arguments as a string

```
kobe@kobep1:~ $ ps aux | more
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.5  0.2 33832  8080 ?        Ss   05:34   0:04 /sbin/init
root         2  0.0  0.0    0     0 ?        S    05:34   0:00 [kthreadd]
root         3  0.0  0.0    0     0 ?        I<   05:34   0:00 [rcu_gp]
root         4  0.0  0.0    0     0 ?        I<   05:34   0:00 [rcu_par_gp]
root         8  0.0  0.0    0     0 ?        I<   05:34   0:00 [mm_percpu_wq]
root         9  0.0  0.0    0     0 ?        S    05:34   0:00 [rcu_tasks_rude
_]
root        10  0.0  0.0    0     0 ?        S    05:34   0:00 [rcu_tasks_trac
e]
root        11  0.0  0.0    0     0 ?        S    05:34   0:00 [ksoftirqd/0]
root        12  0.0  0.0    0     0 ?        I    05:34   0:00 [rcu_sched]
root        13  0.0  0.0    0     0 ?        S    05:34   0:00 [migration/0]
root        14  0.0  0.0    0     0 ?        S    05:34   0:00 [cpuhp/0]
root        15  0.0  0.0    0     0 ?        S    05:34   0:00 [cpuhp/1]
root        16  0.0  0.0    0     0 ?        S    05:34   0:00 [migration/1]
root        17  0.0  0.0    0     0 ?        S    05:34   0:00 [ksoftirqd/1]
root        19  0.0  0.0    0     0 ?        I<   05:34   0:00 [kworker/1:0H-k
blockd]
root        20  0.0  0.0    0     0 ?        S    05:34   0:00 [cpuhp/2]
root        21  0.0  0.0    0     0 ?        S    05:34   0:00 [migration/2]
root        22  0.0  0.0    0     0 ?        S    05:34   0:00 [ksoftirqd/2]
root        24  0.0  0.0    0     0 ?        I<   05:34   0:00 [kworker/2:0H-k
```

- **Processes Difference Before and After Restart**

- In comparing the processes *owned by me* before and after reboot, I noticed that the processes resulted in no difference. There were still 6 processes running under my own user ID. The process IDs were all different, however, the same command was still shown for each of the processes. The reason for this result is due to me not having executed anything before the restart, and after the restart not having done anything at all.
 - *Process Files: myprocess1.txt and myprocess2.txt*
- In comparing the processes *owned by root* before and after reboot, I noticed that there are more processes running after the reboot in comparison to before. The reason for this is most likely due to processes starting up and spawning other processes to complete its initial tasks. In contrast to before the reboot, the system has already completed all its initial tasks and has been idle for a while.
 - *Process Files: rootprocess1.txt and rootprocess2.txt*

3. [Programming Problem] The objective of this exercise is to implement a multithreaded solution to find if a given number is a perfect number. N is a perfect number if the sum of all its factors, excluding itself, is N ; examples are 6 and 28. The input is an integer, N . The output is true if the number is a perfect number and false otherwise. The main program will read the numbers N and P from the command line. The main process will spawn a set of P threads. The numbers from 1 to $N-1$ will be partitioned among these threads so that two threads do not work on the same number. For each number in this set, the thread will determine if the number is a factor of N . If it is, it adds the number to a shared buffer that stores factors of N . The parent process waits until all the threads complete. Use the appropriate synchronization primitive here. The parent will then determine if the input number is perfect, that is, if N is a sum of all its factors and then report accordingly.

- **Program stored in folder Q3**

- *Requirements: Python 3.2 or newer*