Lab 2

Introduction to Scripting

Outcomes

By the end of this lab, you will:

- Have written an initial set of short shell scripts that illustrate the use of utilities and loops.

Rubric

- Each script has specified points for completion of either the script or individual components of the script. The scripts need to be checked off in lab and turned into iLearn in order to receive credit.

Topic: Shell Scripting

UNIX System administrators write shell scripts to automate repetitive tasks. And, almost anything you do more than once with a computer can be automated! A general rule of thumb is to automate anything you will have to do exactly the same way three or more times. This includes things like:

- Installing computers.
- Modifying configuration files on servers so that they perform a particular role; eg. as a web server.
- Checking for operating system and software updates.
- Modifying system state (eg. renicing processes, changing filesystem permissions, monitoring system health).

Thankfully, shell scripting makes these tasks easy. Shell scripting uses an amazingly terse and simple syntax to perform complex operations; this reduces the overhead of writing the software necessary for automation. For some quick insight into what makes a language suitable for this sort of scripting, take a look at the difference in the amount of code it takes to call an external program and check the return value in different languages:

http://merd.sourceforge.net/pixel/language-study/scripting-language/#system

44 characters in sh, 446 in Java... more than an order of magnitude difference in the amount of typing. The fewer the number of characters to perform the task, the faster it is to set up the automation.

So, let's dive in. If you either need a refresher or haven't coded much in bash before, here are some tutorial to get started with things like variables, conditionals, loops, file globbing ,etc.

http://www.panix.com/~elflord/unix/bash-tute.html
https://www.shellscript.sh/first.html

At this point, you should be ready to write some shell scripts. These scripts should be written as tersely as possible. At the same time, they're each going to do things that would take many more lines of code to write in other, less “scripty” languages. Once you've written these scripts, demo them all at once to the TA. If you don't finish either of them by the end of the lab, demo what you do have at that point.

Part 1

Write a script that prints out the number of lines of code with the word “magic” on them in the Linux kernel source tree for the kernel running on your system, regardless of whether the word is upper case, lower case, or some combination thereof. Only count instances in files ending with a '.h' extension.

Hint: this will require installing the kernel-devel package first and finding out where it put its files. It is easiest to figure out which files to look in by using the 'find' command to find filenames that end in '.h'.

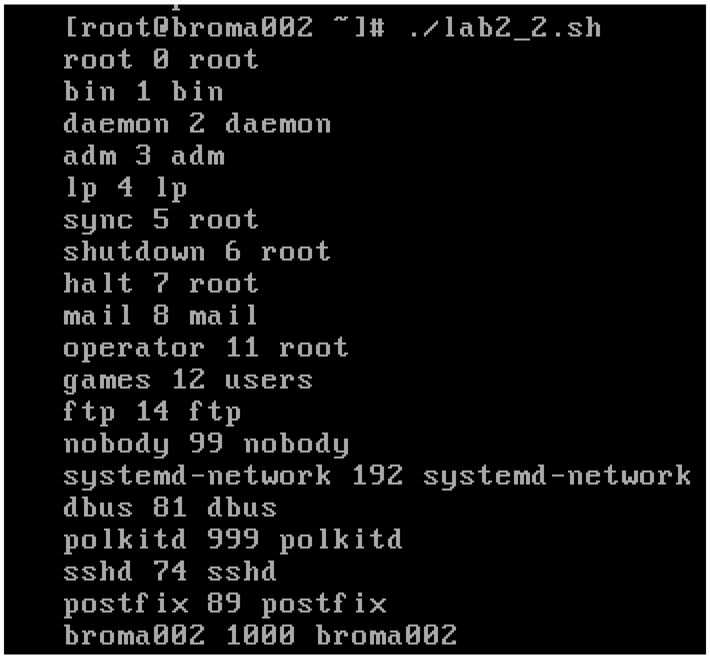
(3 points)

Part 2

Write a script that enumerates the system’s users and groups from /etc/passwd and /etc/group. For each user, print the user’s login, UID, and the groups of which the user is a member. Hints: One way to implement this is as a nested for loop similar to the one in the example scripts that you have access to; using awk with a colon as a separator will help in parsing the password and group files.

The output should be of format similar to:

```
vhill 101 group1 group2 group3
```



```
[root@broma002 ~]# ./lab2_2.sh
root 0 root
bin 1 bin
daemon 2 daemon
adm 3 adm
lp 4 lp
sync 5 root
shutdown 6 root
halt 7 root
mail 8 mail
operator 11 root
games 12 users
ftp 14 ftp
nobody 99 nobody
systemd-network 192 systemd-network
dbus 81 dbus
polkitd 999 polkitd
sshd 74 sshd
postfix 89 postfix
broma002 1000 broma002
```

(3 points)

Part 3

Write a script that will continuously monitor a local file and write to a log file depending on the success of that operation. If the program does not receive a successful file check it should add an entry to a log file `/var/log/cs183/uptime.log` of the format:

```
04-09-19 15:31:55 - File "/full/path/to/lab2-test" has been lost
```

Note that this should only be logged the first time the script triggers and the file is lost (or if there is no existing log). If the last log was that the file was lost, and the file is still lost, nothing needs to be logged. This log file should next be updated once the file can be read again with the time that the success occurred with a line of the format:

```
04-09-19 15:33:55 - File "/full/path/to/lab2-test" has been found
```

Like when the file is lost, this should only log once when the file is found and as long as the file is can still be read no other logs should be written (until it becomes unreadable again). The script should run at an interval of once every 2 minutes to perform these checks, and you can setup this automation via either crontab or systemd timers.

Extensions that you can create on your own time.

- Extend the script to monitor a list of files.
- Extend the script to monitor uptime of a server/domain.

(4 points)

Submission

Create a tarball which contains the three scripts that you created as well as the systemd unit and timer or crontab file and submit it to the Lab 2 iLearn submission.