Andrew Baca
February 8th, 2019
CS480
Assignment # 3

1.1) **at**
The at command is used to see which jobs or commands at a specified time in the are executed in the future. The time specified to run the command usually is in the form of HHMM, HH:MM, as well as an array of keywords such as midnight, noon, or teatime, And the date that the command is to be executed can be formatted in various ways including DD.MM.YY, MMDDYY, and MM/DD/YY. The superuser, or root, can execute this command at anytime, however, other users may be determined in the PERM_PATH.at.allow file or the PERM_PATH.at.deny file, and emails will be sent with the standard output and standard error.
If yo are specifically worried about the times that the processes are running, and not the processes themselves, you may want to use the -v command, which will print out the times before the process names.

1.2) **cron**
The cron command searches for crontab files and loads them into memory to be examined for any jobs that need to be ran within the minute. Whenever commands are executed, the owner of the crontab file will be notified via email with the output. The crontables can be checked by checking on its last modification time using modtime. Some common files and directories that the cron command checks contain cronjobs and crontables.
You would want to use the -m option followed by a mail output if you want to check if your cronjob is running. This can be useful if you are away from your system, but still want diagnostics of what is running.

1.3)**crontab command**
The crontab command is used to access, install, and maintain the various crontab files for a specified user. Each user has their own crontab file, although the root can control access to these with the cron.deny and cron.allow files. When the crontab command is called with certain options, you can access and modify these crontab scripts from user to user.
If you are an administrator, and you want to stop a process that a user is constantly running to save resources, you may want to use the -e option followed by the -u option followed by the certain username, and comment out or erase the job that is hogging resources.

1.4) **crontab config file**
The crontab config file is used to declare commands that you want ran on a set schedule, as well as the time intervals that you would like them ran at. The format of the cron config files accepts 5 fields for time units in minutes, hours, days, weeks, and months, followed by what command or script that you would like to be ran every time that was declared.
Since this is a config file, to access this file, you would use the command crontab -e, and the option -e will pull up the crontab file specific to the user who called it, if they have permissions to use cronjobs.

1.5) **date**
The date command will display the system date and time when it is called. Different options and format control interpreted sequences are used in order to filer the results of time that you want displayed, affecting format, such as full names opposed to numbers, days of week, and even what century that we are in.

If you were a teacher, and wanted to see the last time that a student worked on a certain file, you may want to use the date command with -r option, and what this will do is display the last modification time of a file.

1.6) **fuser**

the fuser command is used to show the user which processes are using what files, sockets, or file systems. When called, the fuser command will display the process ids using the file specified by the fuser. If no argument is given to fuser, then a summary will be displayed with a list of options that will filter the results of fuser.

If you noticed that there is a file or directory that you do not trust, and many processes are accessing the file, then you can suspend the processes communicating with this untrustworthy source using the fuser command with the -k –kill option to kill all these processes while you investigate further.

1.7) **lsof**

The lsof command is used to list information about various files, directories, and libraries that are currently open in the system. When lsof is called, it will list all of the currently open files, along with certain attributes such as all of the processes that these open files belong to, what kind of file or process they are, and the source path.

If you do not know the exact ID's of all the users, but you know all of the users by their usernames, you may want to use the -l command so you can see a list of their usernames opposed to their ID's.

1.8) **pkill**

The pkill command is used to signal processes, using options along with patterns in order to match criteria of the processes to signal to. The patterns used to match the process criteria comes in the form of a regular expression. Some of criteria used to match the regular expression might be PID, process names, group ID, time it started, etc.

If you are interested in getting the name of the process as well as the process ID, you would want to use the -l option, which will list the process name as well as the process ID. If you are only interested in processes belonging to groups, you can use the -g command to filter out other processes.

1.9) **ps**

When the ps command is called, a table is displayed filled with information pertaining to active processes. The ps command is not real time, but rather displays the information once at the time called. The ps command comes with a whole array of options, which can be mixed together, and are used to vary what is displays, such as different users different processes for instance.

If you are specifically interested in seeing which permissions the users have running these active processes, you would want to use the -user option to print the effective user ID of the users, pertaining to permissions.

1.10) **top**

The top command can be used to print a summary if the system in real time, updated every couple seconds. Some of the columns of information being displayed when top is called are list of processes, users of the processes, and resources being used such as memory and CPU percent.

When using the top command, one option that you might want to use to get a mare detailed view of resource use by using the -H option to see the thread usage in each process.

1.11) **umask**
The umask command is used to set a mask if you want to use permission bits only. This command is commonly used in system calls that create files or directories, to set a standard of permissions. Whenever umask is called, a number will be displayed in the terminal, corresponding to the previous value of the mask. Above the list of tables, is an overall summary of the system, displaying total tasks, their state, and all the resources used and free.
You can use the -p options if you want to omit mode, and then the output form is reused from input

1.12) **uptime**
The uptime command is called to display system information pertaining to system times, users, and jobs in the sytem. When called, a table is displayed showing the time called, how long the system has been up, how many users there are on the system, and the average number of jobs put in the run queue in the past 1, 5, and 15 minutes.
There are two options that you can use with the uptime command and they are –help and –version, which are basically just the basic options that all commands have.

1.13) **w**
The w command is used to see information about what users are logged on to the machine, and the processes that they are running, displaying time, how long it has been running, number of users logged in and system loads. The JPCU column shows the time used by all of the processes, and the PCPU column displays time used by the current process.
One option that you may consider using while running the w command is the -s option, which will print out a shortened format, still with the necessary diagnostics pertinent to users and load averages.

2.a) in order to recognize that a process is hogging all the resources, you would want to use the top command, which will summarize what resources are being used where. You can use options such as -o to get certain field names in order to see which process is using the most, such as memory or CPU.

2.b) We can use the kill command along with the -STOP option, followed by the PID to temporarily suspend the processes activity while we investigate the credibility of the process.

2.c) If we later find that this process is credible and it was a process being ran by our boss , and would like to resume its execution, we would later call the kill command with the -CONT option, followed by the PID to resume the process.

2.d) If we need to make sure that the process is dead and cannot resume running, we must use the kill command followed by the -KILL command and the PID to completely terminate the process. If we want information saved in the core directory, we would use the -QUIT option, so the core dump information will be saved there.

3)
In order to disable a user’s privileges that has been abusing his crontab privileges by constantly running expensive tasks, I did a few things.

First, I logged into user jim (username: jim password: jim12345), and used the crontab -e command to pull up the crontable.   From here, I added a command that would execute every minute.
Next, I checked the mail located in /var/spool/mail to see that the command in the crontab was in fact executing every minute.
Once I saw that the task was running properly, I logged into root, and went to /etc/cron.deny. From here, I erased guest, and added jim to revoke his crontab privileges.
In order to fully remove the crontab command that jim was calling every minute, I called crontab -e -u jim to access jim's crontable, and commented out his command.
In order to make sure that this worked properly, I logged back in to jim and called crontab -e, where I was denied permission. I also checked the mail to make sure that the command wasnt running anymore.

4)
There are 4 different time elapsed directories providing another way for software packages to install periodic jobs that I found it the csvm. They are /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, and /etc/cron.monthly.
Three of the directories were empty and didn't have any scripts in them by default, however /etc/cron.daily had three different scripts in it. These scripts were named mdadm, suse-clean_catman, and suse-do_mandb.
Within the first script, mdadm, I first observed that variables were getting set to mdadm sbin and sysconfig file path, so this script was using mdadm config files. The mdadm commands manage md devices, also known as RAID. There is an array of conditional statements in this script MIAL, CONFIG, SCAN, and PROGRAM. The eval command is also used which is a bash builtin command, which concatenates a bunch of mdadm properties together.
Upon reading the contents the suse-clean_catman script, you see that there is a umask command followed by 022, which sets the permission bits for this script. Next thing I observed is that PATH was being set to various bin and sbin directories, and used as an argument with the EXPORT command, and a condition that deleted obsolete man-pages using commands such as test and find.
The suse-do_mandb script looked similar to the suse-clean_catman script, where there is a umask call at the beginning to set permissions, followed by PATH declarations, and tests with man pages.
Overall, it looks like the daily scripts that are ran within the cron.daily and get started from the cron_run file.

5)
I created a Perl script that determines the total VSZ and RSS of all of the processes running on the systems. This was done by using the ps aux command and piping the result using the awk library so that only fields five and six were stored, representing VSZ and RSS. Next, I used a while loop to add the VSZ and RSS into the total VSZ and total RSS variables. When the while loop was complete, I outputted the total VSZ and RSS usage.
In order to compare to the actual results of physical memory and swap space, I used the top command and noted the swap and memory sections and compared them to the VSZ and RSS results that my script outputted.
The output can be found in the tar file in the textfile prob5output.txt.
What I noticed is that the VSZ is the virtual set size, and it is the amount of resources that are allocated to a process, while RSS is the resident set size, and it shows the resources that the

process is actually using at the moment. You can compare these numbers to the actual numbers using the top command.

6)
I first began by creating a shell script, and adding the date command to it in order to first print out the date and time that it executed. Next, I added the w command so that the amount of users and the load would be printed under the date. Then, I added the ps aux command so that a list of all processes will be printed along with the users that they belong to.
After I tested the output, I fed the output to the /tmp/state.log file, and I created a cronjob to run every five minutes that will execute my state.sh script.
I verified that all of this was working by checking the state.log script after a 20 minute span to see how many times the information was outputted.

7)
In order to copy the /tmp/state.log to the /tmp/statelog.YYYYMMDD file, I used the command *cp /tmp/state.log /tmp/statelog.`date +%Y%m%d`* which copies the contents of state.log to the statelog.YYYYMMDD file using the date command with the specified format to print YYYYMMDD.
Next, I used the find command to remove all files in the source directory /tmp/ with the filename statelog.* if they were older than 14 days, so basically, after a certain point in time, there should always be 14 statelog files, with 2 weeks of diagnostics.
Note: to test if the delete was working properly, w set the find command to delete day old files to see if it was working. The command looks as follows:

```
find /tmp/statelog.* -mtime +1 -exec rm {} \;
```

Lastly, I added the rm command to delete the state.log file, so a new one will be executed in 5 minutes. :

8)
In this question, I made a command in a shell script check_arg.sh to check if a valid option is supplied, followed by a username, or multiple usernames. If a valid option is supplied with the username, the username is checked against a regular expression to see if the username is valid. The option -i represented interactive mode, while the option -n represented non interactive mode.
I began by making a base case to see if any arguments were supplied at all, and if there were no arguments were supplied, there was a usage message that was to be printed out. To check the arguments in most cases, I used an if conditional statement and compared the arguments with the targets. I used $# to get the total number of arguments. I used $1 to $n to to get the arguments from index 1 to n.
Next, I checked if the first argument supplied was the -i option. If so, I checked if there was exactly two arguments provided. If so, then if the second expression matched the regex, then I printed the correct diagnostics. If the first argument was -i, and there are more than 2 arguments supplied, I made sure the second argument was valid, and went on. The other arguments were checked with a for loop later in the code.
I did the same process for the -n option that I did for the -i option, checking cases for exactly 2 args, and then the second arg if more than two were supplied.
Finally, I had a for loop at the end that evaluates the arguments 3 to n, n representing total arguments supplied, to evaluate the other usernames with respect to the regex and printing the appropriate diagnostics.
The proper diagnostics can be found in the check_args_list.txt file found in the tar file.

Feedback/Breakdown:

TIME SPENT:

1)	3 hours
2)	30 min
3)	1 hour
4)	2 hours
5)	1.5 hour
6)	45 minutes
7)	45 minutes
8)	3.5 hours
cleanup/testing)	2 hours
documentation)	1.5 hours
TOTAL:	**16.5 hours**

Once again I found this to be a time consuming, yet rewarding assignment. I have a whole new set of Linux commands that I understand, and I am far more comfortable with shell languages than I was prior to this assignment. I am glad I started early, because it will give me plenty of time to cleanup, and modify my work.
Automating task was a bit tricky at first, but it is really straightforward when you do it a few times. Revoking cron privileges is really easy with the cron.deny folder. Writing shell scripts are really awesome because you can use commands with them, and parse them with awk.
Some confusing task involved #4, where we had to explain the daemons and functionality of the scripts in cron.daily. After the completion of this assignment, this was still a confusing point to me.