Collin Gros
10-05-2020
cs480

HW 4

PART 1

at lets you specify a command to run at a later time. This is useful for when you are waiting for some kind of event to happen, and you need to collect data from that event. For example, if you wanted to collect some log data 30 minutes from now, you could use *at*.

The *-r* option allows you to remove a previously specified at-job. You have to give it the at_job_id that it was scheduled and it will remove it from the job queue. You want to use this if you screw up an *at* command, or change your mind about running the command at the time you had specified.

cron is similar to *at*, but reads a crontab of a user to decide what commands to execute, and when. *cron* is a daemon started with init.d. It reads from /etc/crontab or /etc/anacrontab to get cronjobs from users' crontabs.

The *-s* option changes where cron sends job output. If sendmail is not installed or mail is disabled, *cron* will send job output to the syslog file. This is helpful for when you don't read or use mail, but instead look at syslog. This might also be useful with scripting, as you can parse the syslog file instead of searching through mail.

crontab (config file) is the file that holds all the scheduled jobs and times for them to be executed. A crontab file can be used to set jobs and times for each user. You can also change environment variables or environment settings inside of this file.

date shows you the current system date/time, or allows you to set the current system date/time. You can specify the format of which you want the time displayed. This is helpful in scripting, so that you may grab the specific part of the date string that you want, and use that to make modifications to the system.

The *--date* option lets you pick the format described by a given string, instead of the format specifiers. This is helpful if you want to quickly get the date format, instead of having to specify exactly what you want using all of the escape sequences. For example, *--date="next Thursday"* gives you the date for next Thursday.

journalctl queries the systemd journal for you. You can give it output specifiers to narrow down what you would like *journalctl* to give you from systemd. It requires root permissions to view the

file, however. This is helpful if you want to look at the systemd journal for a specific field, or event that happened.

The *-o* command lets you specify the specific output format that you want, among some predefined formats. For example, you can do *-o json*, and the systemd journal will be output as JSON data structures.

ps displays current processes. You can specify different options to narrow down what you're looking for, or ask *ps* to display every process under every user on the system. You can also specify how you want the output to be formatted. The output format options are helpful when you want to save the output of the processes in a certain format. The *ps* format options are helpful for showing only the processes that satisfy certain conditions, so that you don't have to look through every process on the system to find the ones you want.

The *-e* option displays all processes under the current EUID. This is helpful for looking at only processes that you own, especially when on a system that does not grant you any special privileges.

The *-p* {pidlist} option lets you specify the processes that match the PIDlist you give it. This is helpful when you know the PIDs of the processes but want to learn more about them.

sleep pauses for the specified number of seconds. This command is extremely useful wherever you need to wait on another process, or wait for some kind of change to occur. You can also pause for a number of minutes, hours or days.

You can use *sleep* with a suffix to switch between seconds and minutes, hours, or days. This is helpful when you don't want to calculate how many seconds are in a long period of time.

systemctl is used to control systemd, which lets us issue commands to manage services. You can issue unit commands, unit file commands, machine commands, job commands, environment commands, manager lifecycle commands, and system commands.

The *--failed* option isolates all of the units in a failed state. This is useful for debugging.

The *--runtime* option temporarily makes changes that are lost on reboot. This is useful if you're messing around with systemctl and are not completely sure what you're doing.

systemd is a manager for system services. It is run at startup, and initializes every other task by starting services. Dependencies are managed with unit file types that can be configured for

running services. It responds to many different kidneys of signals; with SIGTERM causing it to execute exit.target (which exits systemctl).

The *--crash-shell* pulls up a shell whenever systemd crashes. This is useful for fixing broken configurations whenever systemd crashes.

systemd.service configuration files are files that store information about processes that are controlled by systemd. Three different sections are used in these files: Unit, Install, and Service. Unit and Install are the commonly required sections

trap traps signals. Given an action and signal, trap will execute the action when trapping the signal. This is extremely useful for cleaning up programs before exit or providing different functions depending on the trapped signal.

If the action is -, the shell resets each condition to its default value. This is useful when you want each condition reset.

uptime reports the runtime of the system. It can also give you the time, number of users, and average num of jobs in the run queue. This is very useful for making determinations as to what kind of state the system is in, and if it is safe to poweroff or reboot.

w reports information about logged-in users, as well as *uptime* details. Using this command, you can see what user is logged in and from what host. You can also see details about JCPU and PCPU times, which is useful for system administration.

The *-h* option omits the header. This is helpful for scripting when you want to just parse data that doesn't include system load/uptime information.

PART 2

The /etc/shadow file is for storing user password information. The hash of each user's password is stored next to their name, as well as the algorithm used. This information can give unprivileged users good information regarding the password of another user. This isn't as secure as hiding the hashed password, as hiding it requires more work to discover the password. Password expiration dates are also stored here, which is useful for maintaining network security on a host with many users and incoming traffic.

PART 3

For this part, I used *find* to only give me the inode of every file under a given directory. Then, I use *uniq* to get all duplicate inodes (meaning they are hardlinked). After that, I used perl's split

function to allow me to iterate through every inode found, and finally used a *find* command that executes ls to get the desired output.

This part was the easiest part for me.

PART 4

I use head with w to get the current load, and use tail with w to get the number of users logged in. I specify a custom format for ps to write to state.log. I also edited the crontab to make it execute every 5 minutes.

This part was easy for me as well.

PART 5

To move the log, I store the date in a variable and then append everything in /tmp/state.log to the new file. I check if any logs in that directory are older than 2 weeks by adding 2 weeks to their date and comparing the seconds of that date to the seconds of the current date. If the current date is longer than the 2wks date, then the file is removed. I edited the crontab to include this one as well.

This part was the hardest for me. It took a long time to figure out how to compare the dates (and I realized date had the *-d* option a near the end, which was causing most of my trouble).

PART 6

I just followed yasts' prompts for this part. This was easy.

PART 7

I saw from the lecture notes that *systemctl set-default graphical.target* would accomplish this. This was easy.

PART 8

I use awk a couple of times to isolate fields. I do some string processing by using ps combined with tail to get the process name, and pwdx to get the process path. $$ grabs the PID.

This part was a good challenge. It was helpful to follow the hint given in the lecture notes.

PART 9

I edited /etc/systemd/system/trap_test.service to include the description in [Unit] and the executable in [Service]. Then, I ran *systemctl add-wants multi-user.target trap_test.service*, as well as *systemctl add-wants graphical.target trap_test.service*. Finally, I did *systemctl enable trap_test.service* (which I don't think worked), and then *systemctl enable trap_test* and *systemctl start trap_test*.

This part was fairly easy to do and didn't take me very long to do it.

SUMMARY

I didn't have too much trouble with it all except with part 8 and part 5. I had to take a break somewhere in the middle because staring at code was getting me nowhere. However, coming back, I was able to read more man pages and figure out how it all went together. This assignment took be roughly 7 hours to complete.