Αλέξανδρος Συμπέθερος
AM: 1115200900261
Προγραμματισμός Συστήματος
Άσκηση 2

README

Run on personal computer: Ubuntu Linux (10.10,12.04) and all Linux machines in the Unix Lab(linux12, linux13 and a few others are at this date currently disconnected)
Code is 100% my own.
Helpful sources for coding problems: manpages and various unknown blogs for sed/awk help,hints and ideas.
Code written in vi/pico/gedit but tested on the linux machines of the unix lab.(Solaris machines don't allow last -x and a few other commands(problems with bc..))

About the code:
-Inline Arguments:
When user runs program, correct arguments must be given or script will print an error and exit.
User may enter arguments in any order, but arguments that need extra arguments must be given or error message and termination will occur.(For example: -l-d di.uoa.gr, script expects atleast 1 machine after flag -l, and before next flag).
Also, user may run only with one of the two: -l, -f and accordingly only one of: -s,-st,-c.
Also, user cannot give -t, unless -c is available.
As states the exercise, depending on which flag is given(-l -f, or none), an array of machines is created, for the various outputs to work on.

About Output1:
For every machine in the array, a "finger" is called via rsh, catching all the currently connected users. Obviously, after finger is called, the result is run through a few pipes to get the exact data we need. This piping is true for almost every utility used in the code.
For every user found in that machine, another more specific finger is called(finger -m user) and all the necessary data is retrieved.
Also, for the given user, a "ps" is called, with a few arguments
-(r: gives all running process because ps normally shows dead or paused processes)
-(-U is enabled so ps will only show process for given user)
-(-o changes with output columns ps will show, in this line ps shows cpu of each process and comm with is the process name(without arguments))
From each finger, the script also retrieves all the data of the current user's logged in times(On since...), this data is stored in a file that is given to the function "calculate_date"
Without going into details(the code is very c-style), this function reads the file, retrives the dates, converts them to seconds, compares and finds the smallest(which means the oldest date) and returns the smallest date accompanied by the corresponding host.
This data is retrieved for every user of the current machine and printed to the screen in a nicely manner.

About Output2:
Output2 starts a loop that never ends, unless a control+c command is given.
In each loop, another loop runs that calculates some data for every machine in the original array.
After all the calculations and actions, depending on the "t" , a sleep is called for "t" seconds. The "t" parameter is by default 10, but can be changed when running the script the first time with "-t ?".
Inside every loop of the machines, a finger is called for that machine to retrieve all the users. Also,

besides the user, a “w” is called for the given machine to retrieve the upload times of that machine. Because the outside loop never stops(theoretically speaking), the inside loop must collect upload times for all machines, allowing the user to press control+C and get the 20 most recent upload times, so, when more than 20 upload times have been collected, a simple sed is called that removes the oldest line in the file for the specific machine, moving the total down to exactly 20 lines.
After wards, all the users read from finger are stored in an array uniq_array(which collects users of all the machines) and when all the machines have been passed through, using “sort” “uniq” “wc -l” the total number of uniq users are counted.
Besides the uniq users and the average per machine, a function “ calculate_average” is called that will create the two files necessary for creating the gnuplot, the .plt and the .dat file.
After adding the standard lines to the .plt file, the function loops through all the machines in the original array, and using a file created in the main section of the code “ (pc name...).average” that has the last 20 most recent load times, calculates the average for each of the three columns.
When calculations have finished and a few more lines are added to the .plt file, a gnuplot is called that creates the .jpeg plot.
These actions continue to happen, in essence creating jpegs, until control+c is issued. When this happens, a tar ball is created in the users home file, with the directory that holds all the jpegs previously created. In the end, exit 115 is called

About Output3:
For every pc in the array, call “last -x” (x exists in order to catch the shutdown lines).
Stote all this information in a file called last_file.
3.1:
After removing all the unnecessary data(root, reboot, shutdown...), get all the uniq user from the last_file, and for each user add all lines in the last_file into another more personal file (user)_log. Using that file, call a function “calculate_user_info” that runs through all the lines(which are the “last” results for that specific user) and calculate total time connected to the system.
After collecting all the user's total times, users are sorted based on total time spent, and the top 100 are sent to the report.
3.2:
Because of the -x flag , when last -x called, the log file will contain records of all the reboots and shutdowns happened. With a simple grep, a few temporary files, and an awk, the total number of reboots(and shutdowns) are calculated and all days of that happening are added to the report file.
3.3:
When the function “calculate_user_info” is called, using a simple but powerful awk, the total times a user has connected with a given host, is grouped together and counted.
3.4:
The first thing done, is the current date minus 10 days is calculated, allowing the script to remove all entries in the last file that are older than 10 days.
Using the new file created that has entries for the last 10 days only, a function “calculate session” is called. This function does a few things.
First, it awk's the modified last file(last 10 days), and shows for each date the total number of sessions. Specifically, the modified file has the last few columns of last, the date, the time of logging-out and the (:). The awk function checks the day from the date and creates a file that has the total number of occurrences for each day.(This works because the last 10 days can never have the same day...). Afterwards, a sort and head is called to the above results, in order to keep only the

Αλέξανδρος Συμπέθερος
ΑΜ: 1115200900261
Προγραμματισμός Συστήματος
Άσκηση 2

README

top 5 days with the most sessions.
Because the days we have are numbers(1 to 30), a separate file is kept that converts the date back to its original(day:1 becomes Tue 2 May for example).
Also, using regular expressions and sed, a file is created for all the above dates, containing all records for each day.
As in the ouput2, all necessary information is passed to the .plt file.
Now starts the core loop, which runs through all the days chosen(best 5 found previously).
For each day, after passing to the .plt the necessary information, and adding to the .dat the number, the total number of sessions, another loop is started that tries to find the largest session of that day.
When the second loop finishes, results are stored in .dat file and the outer loop continues.
In the end, a full .plt file is created with xtics labels created correctly and a .dat file that has records:
number_of_day number_of_sessions largest_session
And by using the xtic label mechanism, the day(for example 1, is plotted as Tue 2 May).
3.5:
After all report files are created, a tar ball is created in the home folder or running user that contains the reports. This tar ball is sent via attachment using mutt, as en email to the current user in the current domain.