Dylan Dunsheath

Description (Homework 1)

CS-288-006

Professor Dale

Problem #1

This script was rather simple to incorporate / follow the instructions. The main logic is to get the input from the user asking for the filename and then following various conditions that may be met: did the user provide a filename? Is the 'filename' a directory (done by the flag -d)? Or can it not be found (! -f). After checking if the string/length of filename is NOT 0, it's NOT a directory, and it exists; I I used wc to count the number of lines that the file has. If it's 0, the file is empty. Otherwise, I used the tail -n 10 method to display the last ten lines of the file (tail -n 10 "$fileName).

Problem #2

This script was also rather simple. The first important part is to allow the script to accept command-line arguments (done via "$i" where is the position of the command-line from 1 to n [0 is the script name]). My idea was to do error-checking; like if the user didn't actually type in any input. If they don't, I decided to do a while loop to continuously get input if it is invalid. When the user decides to type in a valid directory/file, it would break out of the loop. Otherwise, if the command-line is valid, it passes the input to userInput. If the input is a directory,, output the directory and then check if the user had read and execute permissions on it. If they do, get the number of files via find and using wc -l to get the number of subdirectories and files in the current directory (checks subdirectories and adds the files in those subdirectories as well). Also, because the directory we are IN is also included in the find command for the subdirectories, we have to decrement it by 1. We then can display the files and subdirectories We then check the total subdirectories and files and if it is greater than 0, we can display disk usage. find "$userInput" -type f -exec du -h {} + 2>/dev/null was used (--exclude was used for subdirectories along with changing ‘f’ to ‘d’ for type). ‘-exec du -h {} +’ is what calculates the disk usage and makes it in human readable format. If there are any error messages, we send it by the ‘2>/dev/null’ (permission denied errors for user for instance). There’s also error-checking mentioning how no files were found and also subdirectories rather than disk usage. We then display a message if the user doesn’t have the permissions if that’s not the case, rather than total_files, etc.

If the input is a file, we displayed it is a file for the user. If the user has read permissions, we got the count of words, lines, and characters and displayed depending on the file if it is “Heavy” “Moderately Sized” or “Light File”.We then displayed if the user had read permissions, the output for words, lines, characters, and the type of category the file is; if they don’t have read permissions, then we just displayed “No read permissions”

Finally, if the input is neither a directory or file, we just said it doesn’t exist and exit the script.

Problem 3

In this script, it is required to have a minimum of 3 files. So I started off by checking if 3 arguments were even passed to the script. If not, it threw an error and showed the required format. Otherwise, if at least 3 arguments were passed, it is good so far. Afterwards, I looped over arguments and verified if it was a directory or a non-existent file. If directory, I threw an error and quit the script since at LEAST 3 filenames are required (directories don’t count!). If a non-existing file, I let the user know it doesn’t exist and we are skipping it. I then used the date() command (using +%Y%m%d to get YYYYMMDD format) and assigned this value to backupDir for later. I then checked if that directory exists before attempting to create it so we can always make backups if needed of the files (if it doesn't exist, make it). The logfile was declared by using the backup directory and then a name called 'backup_log.txt' which is in this subdirectory. I then appended the header of "Original File | Backup File" to the logfile so the user can verify the names of the backup.

Afterwards, I looped again through the arguments and after verifying that it is in fact an existing file, I copied it to the backupDir (where it still has its name we passed) and then got the current timestamp (YYYYMMDD) and added that to the end of the file that's in the backup dir by using the 'mv' command which can rename files. After the final for-loop it notifies the user where their backups are via the absolute path.

Problem 4

This problem was a lot more complicated and overall was a lot more involved unlike the previous three problems. To successfully complete what was required, I took the necessary steps and made functions that would be called (they needed to be called either because an error was met or to do another part that was required). The first thing that is done is the function get_directory_choice is called which allows the user to select any directory they choose. The directories are outputted to the user so they know the directories they can choose from. When the user enters a directory, various things can happen: if nothing is entered, we let the user know that it's empty and call the same function again to select another directory. If a directory doesn't exist, then ask them if they wanna create it; where valid inputs are 'y' or 'n'. Everything else is invalid. I did this via a select due to its simplicity. If they say "y", show their choice then make the directory and let them know it was created; calling the get_directory_choice to select a directory or make another one if it doesn't exist. If they enter "n", show their choice and call the same function. Otherwise, Keep re-iterating until they either enter y/n (since this is in a while true loop). If the user enters a valid directory, cd into it and verify if it doesn't contain anything, otherwise there's nothing else user can do. Give the path to this directory in the function.

Then, call the select_choice for selecting a file (to show 10 lines at a time) or subdirectory (where we recursively check files that were last modified within the last 24 hours). This function contains a while loop that runs indefinitely unless the user "Quits". The function initially shows the current directory and adds the files and subdirectories. PS3="Select…" Prompt numbers the choices in "Choices" array from 1 - … (based on number of files/subdirectories). I verified the input via regex that it is actually a numeric value. If the user "Quits" exit script. If the user chooses a directory, echo out the subdirectory and cd into it. modded_files adds the file paths since we used DFS to recursively go through subdirectories (since the function search_modded_files is called with the directory path we are in). We then check if the array is 0 and > 0. We then output the paths to the files and verified input. If it is, we called display_file or went back to original directory. If input is simply a file, we call display_file. We also ensure input in general is valid.

Search_modded_files was not terribly long. I initialized several local variables: directory we are in/path, the current time time since epoch (seconds) and things like the seconds in a day, the modified time a file was edited/modified and the difference between current time and modified time. If it's less than or equal to 24 hours, add it to array.

Display_file was really simple. I have a local variable 'file' which gets the file we want to display, the total lines in the file, current lines and the lines we want to show. We verify if the total lines is 0 (empty file), where we strip the path and output that the basename (filename) is empty and go back to original directory, and allow the user to select another file or subdir. If that's not the case, we output the first ten lines and increment the current line by 10 and assign it. We continuously check that iour current line we are on is NOT greater than the lines in file (if it is equal or greater, we let the user know and call originalDir). We then continuously ask the user if they want to display more lines. If they do, increment the current line by 1 (don’t assign yet). Tail -n +$((…)) starts at a certain point and then display 10 lines from THAT line. Increment currentLine by 10 and then continue. Otherwise, go back to original directory or reprompt.