Byung Hun Lee

CS 260

Raheja

Assignment 4 Write Up

The purpose of this essay is compare the four different scripts I wrote for the assignment using bourne again shell (bash), korn shell, c-shell, and z-shell. Furthermore I will state what I found to be advantageous in one scripting language but different in others, basically choosing what I found to be the easiest versus hardest. Honestly speaking however, while it was easy to compare the differences between c-shell and every other script, it was harder to compare bash, korn shell, and z-shell as they were all similar.

I will start with the advantages and disadvantages of bourne again shell., or bash. I found bash to be the overall 3rd easiest script, or the 2nd hardest script overall. Although I’ve placed it 3rd it was still easy compared to c-shell. Almost every method or commands I needed to finish the assignment was already known from previous assignment or from a quick online search. In total the bash script required 141 lines of codes (including comments). There was only one command I did not know how to perform and it was performing float point arithmetic. Although most (actually every ATM I know) does not allow interaction that involves cents I did take transfers that involve cents. Unfortunately, bash by default only worked with normal integers without decimal points and that meant that users could not input a floating point number such as 10.50 when desiring transactions lest the script break, or at least under normal circumstances. Thankfully, there was a workaround that allowed bash to work with floating points. Normally I would use a simple $variableA > $variableB command to compare numbers and $variableA + $variableB to perform arithmetic operations with numbers. However since we were dealing with floating point numbers I had to use $(bc <<< “$variableA > $variableB”) to allow floating point comparison and $(echo “amt + $savings” | bc ) for floating point arithmetics. There were several other difficulties involving bash compared to others. For one, unlike c-shell, spacing was very important in bash, one misplacement involving a space would cause the script to throw an error and crash. Furthermore certain control statements had to be used with parenthesis while others had to be used with brackets or the system would be unable to recognize the command. The last problem appeared while the script was running. As instructed the screen would be cleared in certain places, for example when the pin was entered wrong. Should the wrong pin be entered a error message would appear, the screen would clear and the script would ask for another pin. However, to ensure the error message was read I used the sleep command to pause the system momentarily. During this time period should the user enter another number and hold it without pressing enter that number would carry over. For example the correct pin was 111 but should the user enter 1, press enter, and before the screen would refresh enter another 1, but this time without pressing enter, that input of 1 would be carried over to the next screen. But this number would be hidden, the user would not see it. At this point the user would have to enter 11 for the pin to be correctly recognized, entering 111 would give an error message. Other than the above problems I encountered everything else was a plus for bash, or rather easy to do. Bash’s function support improved writability and readability which could not be said for c-shell.

The second script I will go over, and the second one I wrote was c-shell, the script I found to be most difficult to write. C-shell was the most unique of the four script in that the syntax in this script was largely different compared to bash whereas z-shell and korn shell had very similar syntax to bash. For example I had to use the **set** keyword to set a variable and the **then** keyword following a **if** statement *had* to be on the same line as the if statement for the command to be recognized. Another difference was that unlike the other three scripts, c-shell used the **switch** statements instead of case, plus each switch condition had to end with **breaksw** unlike other scripts that used **case** statements. Furthermore the same problem involving floating point arithmetic found in bash was also found in c-shell. I had to use a specific command using command line operation to perform floating point comparison, addition, and subtraction. In addition, the input and clear problem encountered in bash described above also appeared in c-shell. Lastly, there was no easy way to jump from method to method (that I could implement) so everything was in one giant block of code, although the lack of functions did not necessarily make the script harder to write it just made the code look messier with more tabs to be pressed to make the code look neater. Although the script was 137 lines, making it technically smaller than bash, bash had functions so really there was no difference in length. On the plus side spacing did not matter which I found to be a huge advantage. The only other advantage of c-shell that made scripting easier was that every control statement I used only required the use of parentheses, so thankfully there was a lack of confusion between having to use parenthesis for some and brackets for others.

The third program I discuss is z-shell. z-shell, like korn shell, was very similar to bash syntax wise. I used my bash script as a basis for the z-shell with very little changes between the two. Like bash, my z-shell script had to pay careful attention to spacing. Furthermore like the previous two scripts there was a problem with inputting after clearing when running the script. Lastly, just like bash, I had to pay attention to which control statements required brackets and which statements required parentheses. Unique to z-shell, however, I had to use a single equal sign (=) for comparison whereas every other script used two equal signs. This posed a slight difficulty as I was so used to using two equal signs and the new syntax required some time to get used to. There were some positive sides to z-shell. First and foremost, z-shell by default supported float point arithmetic so there was no special command required for variable comparison/addition/subtraction unlike c-shell and bash. Since z-shell also supported easy function implementation, the script was cleaner compared to c-shell in my opinion. Since there was very little difference between the bash and z-shell, the z-shell script also had 141 lines of code.

The last script to discuss is korn shell. I found korn shell to be easiest of the four and most comfortable. Just like z-shell, korn shell was very similar syntax to bash. The only difficulty I encountered while making the korn shell script was that spacing was important just like bash and z-shell. However other than that one problem I found everything else to be better than z-shell and bash, and certainly c-shell. Just like z-shell, korn shell supported float point arithmetic by default but unlike z-shell, control statements could be written in *either* parenthesis or brackets, which made coding easier. Furthermore the weird input and clear problem found in the other three scripts was missing in korn shell. Typing 1 while the script was asleep would carry over to the refresh and the user could see the 1 still there, so the user could avoid any confusion while inputting numbers. Like bash, korn shell also used two equal signs (==) for string comparison which I was used to so that made life less confusing unlike z-shell. Although the korn shell was also 141 lines of code just like bash and z-shell, korn shell was undoubtedly the easiest script to write of the four.

In conclusion, c-shell was hardest in terms of difficulty since it had the largest number of differences compared to the other three script. Bash was difficult due to lack of float point support. Z-shell had a few problems I had to fix but was easier to write compared to bash. Last but not least, korn shell was the easiest in my opinion since it could do what I wanted without requiring specific commands.